Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Matematika – uporabna smer

Tea Robič

Genetski algoritem za problem urnika

Diplomsko delo

Ljubljana, 2002

Kazalo

Program diplomskega dela

V diplomskem delu obravnavajte problem šolskega urnika. Opišite natančno matematično formulacijo problema in predstavite različne možnosti za ocenjevanje kvalitete rešitev. Podajte tudi kratek pregled pomembnejših algoritmičnih metod za sestavljanje urnikov. Izberite si eno od metod in izdelajte program, ki bo z izbrano metodo ob smiselnih omejitvah rešil problem šolskega urnika. Na primerih ocenite obnašanje izdelanega programa. Za osnovno literaturo uporabite zbornike [1], [2] in [3].

Mentor: doc. dr. Martin Juvan

Somentor: Drago Bokal, univ. dipl. mat.

Literatura:

[1] E. K. Burke in P. Ross, urednika. *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29–September 1, 1995, Selected Papers*, *Lecture Notes in Computer Science*, zv. 1153. Springer, 1996.

[2] E. K. Burke in M. W. Carter, urednika. *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT '97, Toronto, Canada, August 20–22, 1997, Selected Papers*, *Lecture Notes in Computer Science*, zv. 1408. Springer, 1998.

[3] E. K. Burke in W. Erben, urednika. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16–18, 2000, Selected Papers*, *Lecture Notes in Computer Science*, zv. 2079. Springer, 2001.

Povzetek

V diplomskem delu obravnavamo genetski algoritem za problem šolskega urnika. Problem urnika definiramo kot dodeljevanje ur in predavalnic danim predavanjem. Pri tem opišemo omejitve, katerim mora urnik zadostiti, da je dopusten, in omejitve, ki so le zaželene. Slednje uporabimo za ocenjevanje kakovosti urnika. Na kratko opišemo metode, ki se uporabljajo za reševanje tega problema. Za reševanje problema urnika uporabimo genetski algoritem. Podrobno opišemo genetske operatorje (prekrižanje in mutacijo). Algoritem implementiramo v programu Kronos. Na podatkih z Oddelka za matematiko Fakultete za matematiko in fiziko naredimo analizo parametrov genetskega algoritma in prikažemo najboljši dobljeni urnik.

Ključne besede: urnik, problem urnika, šolski urnik, univerzitetni urnik, dopustna rešitev, omejitev, kriterijska funkcija, predavanje, srečanje, optimizacija, optimizacijski problem, metahevristike, algoritem, genetski algoritem, generacija, populacija, osebek, genetski operator, prekrižanje, mutacija, selekcija, selekcija z ruleto, elita

Key words: timetable, timetable problem, school timetable, university timetable, feasible solution, constraint, fitness function, lecture, meeting, optimization, optimization problem, metaheuristics, algorithm, genetic algorithm, generation, population, individual, genetic operator, crossover, mutation, selection, roulette selection, elite

Math. Subj. Class. (2000): 90B35, 90C59, 68T05

Comp. Rev. Class. (1998): F.2.2., I.2.8., J.1.

Zahvala

Zahvaljujem se vsem, ki so mi kakorkoli pomagali pri izdelavi diplomskega dela. Posebej bi se rada zahvalila mentorju doc. dr. Martinu Juvanu, ki me je vodil skozi pisanje diplomskega dela, in somentorju Dragu Bokalu, ki mi je pomagal z nasveti o genetskih algoritmih in programiranju. Poleg tega se zahvaljujem sodelavcem pri Virtualni univerzi, ki so mi pomagali pri sestavljanju baze podatkov in vnašanju podatkov za Fakulteto za matematiko in fiziko. Zahvaljujem se tudi Dejanu, ki me je vseskozi podpiral, in seveda mami, ki mi je vedno stala ob strani.

Tea Robič

1 Uvod

Vsaka šola ali univerza se vsaj enkrat na leto sreča s problemom urnika. To je problem, pri katerem moramo predavanjem dodeliti ure in prostore, v katerih se bodo odvijala. Pri tem moramo upoštevati nekatere zahteve, kot so: predavateljem in študentom se predavanja ne smejo prekrivati, v eni predavalnici lahko poteka samo eno predavanje hkrati, predavalnice morajo biti dovolj velike ipd. Poleg omenjenih zahtev (strogih omejitev), ki so nujno potrebne za dopustnost urnika, pri problemu urnika srečamo še t.i. šibke omejitve. To so omejitve, ki so zaželene, ne pa nujno potrebne (npr. urnik študentov naj bo čim bolj kompakten, predavanja naj bodo enakomerno porazdeljena čez cel teden ipd.). Navadno iščemo urnik, ki bi zadostil vsem strogim omejitvam in čim več šibkim omejitvam.

Ročno sestavljanje takšnega urnika je težko in zamudno delo, ki lahko (odvisno od tega, ali popravljamo obstoječi urnik ali delamo novega) traja od nekaj dni do več tednov. Sestavljalec mora upoštevati številne spremenljivke (študente, predavatelje, predavalnice ipd.) in omejitve, ki si lahko tudi nasprotujejo. Zato so si sestavljalci urnikov že zgodaj poskusili delo olajšati s pomočjo računalnika. Prvi poskusi sestavljanja urnikov z računalniško pomočjo segajo v 60-ta leta, ko so univerze dobile računalnike. Od takrat se je veliko število raziskovalcev in programerjev ukvarjalo s tem problemom.

Skupaj z razvojem matematike in računalništva so se razvijale tudi metode reševanja problema urnika. Te segajo od barvanja grafov do zapletenih metahevrističnih algoritmov, kot so programiranje z omejitvami, tabu iskanje, simulirano ohlajanje in genetski algoritmi. Vse najuspešnejše metode zadnjih let temeljijo na metahevristikah.

Genetski algoritmi so optimizacijsko orodje, ki temelji na Darwinovi teoriji evolucije. Ena njihovih najboljših lastnosti je, da se dobro odrežejo tudi v zelo zapletenih pogojih.

Ti algoritmi delajo s populacijo rešitev, ki jo s pomočjo genetskih operatorjev, kot sta prekrižanje in mutacija, razvijejo v novo, boljšo populacijo. To ponavljajo več generacij in pogosto dobijo zelo dobre rešitve.

V diplomskem delu bomo opisali in implementirali genetski algoritem za problem šolskega urnika. V prvem poglavju si bomo najprej ogledali formalno definicijo problema urnika in njegove dopustne rešitve. Nato bomo na kratko razložili nekaj metod, ki se uporabljajo pri sestavljanju urnika.

V drugem poglavju bomo opisali glavne značilnosti genetskih algoritmov. Podrobno bomo razložili algoritem, s katerim smo se lotili problema urnika.

Tretje poglavje je namenjeno opisu programa Kronos in študiju parametrov uporabljenega genetskega algoritma. Na koncu poglavja bomo prikazali še najboljšo rešitev problema urnika za podatke z Oddelka za matematiko Fakultete za matematiko in fiziko Univerze v Ljubljani.

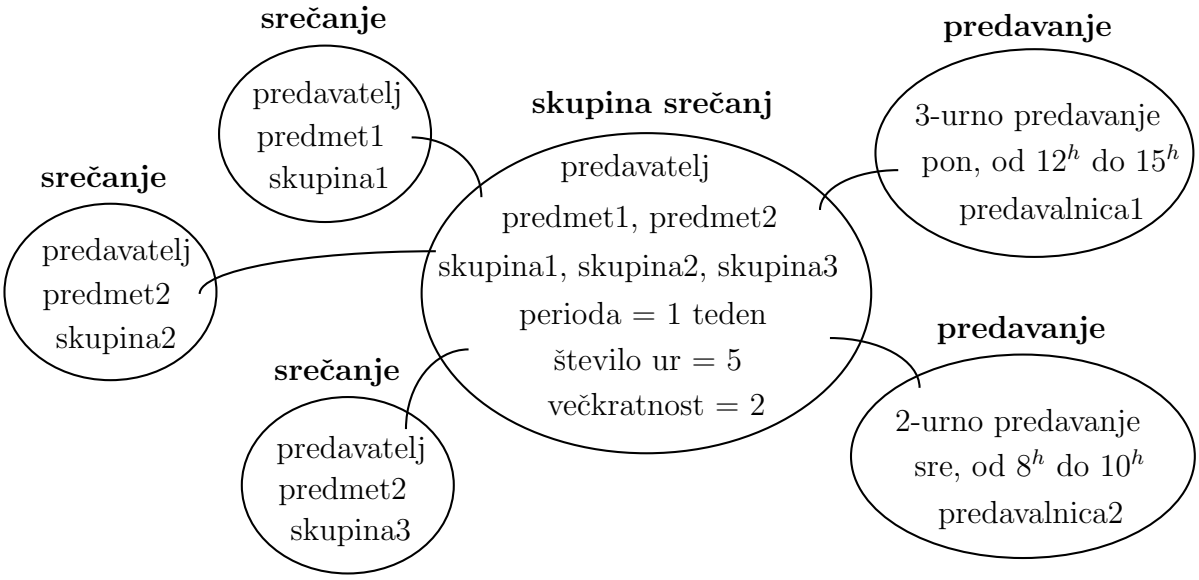
2 Problem urnika

2.1 Opis

Problem šolskega urnika je problem razporejanja predavanj v časovne in prostorske okvire tako, da so pri tem upoštevane nekatere zahteve oz. omejitve. Ko govorimo o šolskem urniku, se srečujemo z naslednjimi izrazi:

- *Predavatelj* je običajno profesor ali asistent, ki vodi neko predavanje.
- *Študent* je obiskovalec predavanja.
- *Razred* sestavljajo študenti, ki so vpisani v isti letnik neke smeri ali študijskega programa. Izjemoma je študent lahko vpisan tudi v več razredov (npr. pri vzporednem študiju).
- *Skupina* je skupina študentov, ki ima isti urnik. Skupino vedno sestavljajo študenti enega razreda.
- *Ura* je osnovna časovna enota.
- *Perioda* urnika je število tednov, po katerih se urnik ponovi. Razdeljena je na ure.
- *Predmet* je del študijskega predmeta. Če je študijski predmet sestavljen npr. iz predavanj in laboratorijskih vaj, sta v jeziku problema urnika to dva različna predmeta. Predmet je lahko skupen več različnim razredom in se v eni periodi lahko pojavi večkrat. Isti predmet lahko predava več predavateljev.
- *Trajanje* nekega predmeta nam pove, koliko ur na teden je po predmetniku namenjenih predmetu in ni nujno celo število.

- *Predavanje* je študijska enota, ki pomeni izvajanje nekega „predmeta" (lahko več predmetov) ob določeni uri v določeni predavalnici.
- *Predavalnica* je prostor, v katerem poteka predavanje. K predavalnicam štejemo tudi posebne učilnice, kot so računalniške učilnice in laboratoriji. Predavalnica je *ekskluzivna*, če v njej lahko potekajo samo predavanja, ki predavalnico izrecno zahtevajo.



Slika 1: Povezava med srečanji, skupino srečanj in predavanji.

- *Srečanje* je šesterica predavatelj, predmet, skupina, perioda, število ur in večkratnost, ki pove, da bo predavatelj predaval skupini študentov dan predmet. Perioda srečanja nam pove, na koliko tednov se srečanje ponovi. Perioda srečanja vedno deli periodo urnika. Število ur pove, koliko ur na periodo bo trajalo srečanje, večkratnost pa, na koliko delov se srečanje v periodi razbije. Več različnih srečanj (katerim morajo biti skupni predavatelj, perioda, število ur in večkratnost) sestavimo v *skupino srečanj*. Ker je predavatelj srečanj enak za vsa srečanja iz iste skupine srečanj, lahko rečemo, da je to kar predavatelj skupine srečanj. (Podobno velja tudi za periodo, število ur in večkratnost.) Skupino srečanj razdelimo na toliko predavanj, kolikor zahteva njena večkratnost. Vsa srečanja iste skupine srečanj se odvijajo na skupnih predavanjih.

Pri sestavljanju urnika moramo predavanjem dodeliti predavalnice in ure ter se pri tem držati omejitev, ki jih delimo na dve skupini:

1. *Stroge omejitve* so omejitve, ki jih moramo pri sestavljanju urnika nujno upoštevati. Urniku, ki se teh omejitev drži, pravimo *dopustni urnik* ali *dopustna rešitev za problem urnika*. Stroge omejitve so:

   - V urnik moramo razporediti vse ure vseh predmetov.
   - Noben predavatelj ne sme imeti več predavanj hkrati.
   - Nobena skupina ne sme imeti več predavanj hkrati.
   - Nobena predavalnica ne sme biti dodeljena več predavanjem hkrati.
   - Predavalnica mora biti dovolj velika.
   - Nekateri predmeti (npr. vaje iz računalništva) zahtevajo, da se izvajajo v eni izmed vnaprej določenih predavalnic. To zahtevo moramo upoštevati.
   - Predavanja se morajo odvijati v nekem časovnem okviru (npr. od 7-ih do 20-ih).
   - Predavatelji imajo lahko nekatere dneve oz. termine rezervirane za druge aktivnosti. Takrat jim ne smemo dodeliti predavanj.
   - Tudi predavalnice so lahko ob nekaterih terminih rezervirane, zato jim takrat ne smemo dodeliti predavanj.
   - Ekskluzivni predavalnici (npr. laboratoriju ali računalniški učilnici) lahko dodelimo le predavanja, ki zahtevajo to predavalnico.
   - Nekatera predavanja so fiksna. To pomeni, da imajo vnaprej določeno predavalnico in uro začetka. Fiksnih predavanj ne smemo nikoli premakniti v drugo predavalnico ali na drugo uro.
   - Predavanja, ki nastanejo iz iste skupine srečanj, se lahko odvijajo le v različnih dnevih.

2. *Šibke omejitve* so omejitve, ki so v dobrem urniku zaželene, ne pa obvezne. Bolj kot jih bomo upoštevali, boljši urnik bomo dobili. Vse seveda niso enako pomembne, zato jim bomo priredili uteži. Primeri šibkih omejitev so:

   - Predmet, ki ima veliko ur, mora biti enakomerno porazdeljen čez celo periodo.
   - Urnik naj vsebuje čim manj prostih ur za študente.
   - Za vsak razred lahko določimo ure, ki so bolj ali manj zaželene za predavanja.
   - Študenti naj se čim manj selijo iz ene predavalnice v drugo.

- Urnik naj čim bolje izkoristi kapaciteto predavalnic.

Nekatere šibke omejitve si lahko tudi nasprotujejo. Taki sta na primer zahtevi, da naj se študenti čim manj selijo med predavalnicami in naj bodo predavalnice čim bolj izkoriščene. Bolj upoštevana bo omejitev z večjo utežjo.

Kot smo že omenili, izpolnitev več šibkih omejitev pomeni boljši urnik. Pri dodajanju šibkih omejitev pa moramo biti previdni, saj vsaka omejitev poveča zahtevnost problema in neposredno vpliva na čas reševanja. Najti moramo torej pravo mero med želeno kakovostjo urnika in časom, ki smo ga pripravljeni vložiti v izdelavo urnika.

Veliko avtorjev se je problema urnika lotilo samo na teoretičen način. Tako so prišli do splošnih algoritmov. Ko pa želimo take metode praktično uporabiti, pridejo do izraza posebne zahteve (navadno v obliki šibkih omejitev), ki se pojavijo v vsakem konkretnem primeru in zaradi katerih je vsaka splošna rešitev žal le omejeno uporabna. Ali kot je napisala Kathryn A. Dowsland v [16]:

Splošen algoritem je kot obleka št. 56. Pokrije vsakogar, a ni prav nikomur.

Tako so implementacije navadno omejene na posamezne ustanove (šole, univerze ali fakultete) in rešitve, ki bi bila dobra za vse primere, ni.

	srednje šole	**univerze**
razporejanje	po razredih	po študentih
izbira	malo izbire, močno strukturirani programi	veliko izbire, šibko strukturirani programi
obremenitev predavateljev	polna	nizka (nekaj ur na teden)
obremenitev študentov	velika obremenitev	manjša obremenitev, ki je razporejena čez cel dan
predavalnice	predavalnice imajo približno enake kapacitete in so na isti lokaciji	veliko predavalnic različnih kapacitet, ki se lahko nahajajo na več različnih lokacijah
kriterij	urnik brez konfliktov	dopuščamo minimalne konflikte za študente

Tabela 1: Razlike med srednješolskim in univerzitetnim urnikom.

Problem šolskega urnika je odvisen predvsem od tipa ustanove, ki ga uporablja. Srednješolski in univerzitetni urnik se namreč med seboj močno razlikujeta (tabela 1). Do največjih razlik pri sestavljanju omenjenih urnikov pride zato, ker je srednješolski predmetnik enak za vse dijake nekega razreda, medtem ko imajo univerze sistem kreditnega študija, pri katerem lahko študenti sami izbirajo del ali celo večino svojih predmetov.

V tem diplomskem delu se bomo ukvarjali z urnikom, ki je narejen za univerzo (natančneje za fakulteto), a ima lastnosti tako univerzitetnega kot srednješolskega urnika, saj je v Sloveniji oz. na Univerzi v Ljubljani kreditni študij manj razširjen kot na tujih univerzah. Razporejanje zato poteka po razredih. Obremenitev predavateljev in študentov je nizka. Predavalnic je malo, a so nam na voljo tako majhne kot zelo velike predavalnice in take z različno opremo (laboratoriji, računalniške učilnice ipd.). Naš cilj je sestaviti čim boljši urnik, ki bo brez konfliktov.

2.2 Definicija

Problem urnika formalno zapišemo v naslednji obliki.

Definicija 2.1. *Problem urnika* tvori urejena 17-terka

$$\mathcal{T} = \langle L, C, G, R, S, \sigma, \eta, \nu, n_P, T, A, B, \zeta, M, U, P, W \rangle,$$

za katero velja:

1. Množica $L = \{l_1, \ldots, l_{|L|}\}$ je množica vseh predavateljev.
2. Množica $C = \{c_1, \ldots, c_{|C|}\}$ je množica vseh razredov.
3. Množica $G = \{g_1, \ldots, g_{|G|}\}$ je množica vseh skupin študentov. Za vsako skupino $g \in G$ obstaja natanko en razred $c \in C$, da je $g \subseteq c$. Vse skupine študentov enega razreda so paroma disjunktne (ne vsebujejo nobenega skupnega študenta) in skupaj tvorijo cel razred. Z $|g|$ označimo število študentov v skupini g.
4. Množica $R = \{r_1, \ldots, r_{|R|}\}$ je množica vseh predavalnic. Z $|r|$ označimo kapaciteto predavalnice r. To je število študentov, ki jih predavalnica sprejme.
5. Množica $S = \{s_1, \ldots, s_{|S|}\}$ je množica vseh predmetov.
6. Funkcija $\sigma \colon S \to \mathbb{Q}$ vsakemu predmetu priredi njegovo trajanje. Za vsak predmet $s \in S$ je $\sigma(s)$ število ur, ki jih ima predmet v enem tednu.

7. Preslikava $\eta\colon S \to \mathcal{P}(R)$ vsakemu predmetu priredi množico predavalnic. Če je $\eta(s) \neq \{\}$, potem je predmet $s \in S$ lahko predavan samo v predavalnicah $\eta(s)$. Če pa je $\eta(s) = \{\}$, predmet nima posebnih zahtev glede predavalnic in se lahko izvaja v katerikoli predavalnici, ki ni ekskluzivna.

8. Funkcija $\nu\colon R \to \{0, 1\}$ vsaki predavalnici priredi vrednost, ki pove, ali je predavalnica ekskluzivna. Če je predavalnica r ekskluzivna ($\nu(r) = 1$), se v njej lahko predavajo samo tisti predmeti s, za katere je r zahtevana predavalnica ($r \in \eta(s)$).

9. Število $n_P \in \mathbb{N}$ je perioda urnika, izražena v tednih.

10. Množica $T = \{t_1, \ldots, t_{|T|}\}$ je množica zaporednih ur, ki so na razpolago v eni periodi. Iz formule
$$n_D = \frac{|T|}{7 \cdot n_P}$$
lahko izračunamo število ur v enem dnevu (oznaka n_D).

11. Množica $A \subseteq L \times T$ je množica parov predavatelj-ura, ki povezuje predavatelje z urami, ob katerih niso na razpolago za predavanja.

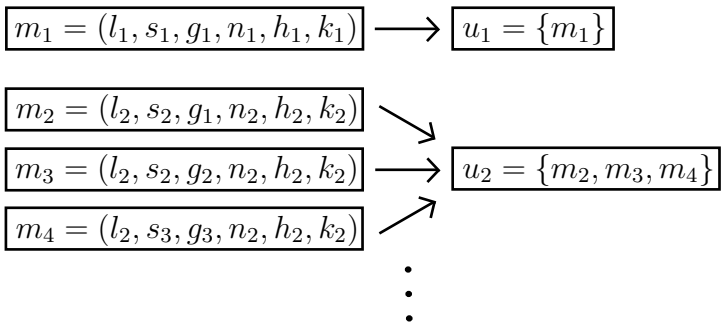
12. Množica $B \subseteq R \times T$ je množica parov predavalnica-ura, ki povezuje predavalnice z urami, ob katerih niso na razpolago za predavanja.

13. Funkcija $\zeta\colon C \times T \to \mathbb{N}_0$ vsakemu paru razred-ura priredi utež. Večja utež pomeni bolj zaželeno uro.

14. Množica $M \subseteq L \times S \times G \times \mathbb{N}^3$ je množica vseh srečanj, kjer srečanje $m = (l, s, g, n, h, k)$ pomeni, da predavatelj l predava skupini g predmet s. Srečanje m se v periodi dolžine n tednov razdeli na natanko k predavanj, ki skupaj trajajo h ur. Perioda srečanja n mora deliti periodo urnika n_P. Če predmet s skupini g predava več različnih predavateljev, mora za vsa srečanja $m_i = (l_i, s, g, n_i, h_i, k_i)$, ki vsebujejo predmet s in skupino g, veljati
$$\sum_i \frac{h_i}{n_i} = \sigma(s).$$

15. Množica $U \subseteq \mathcal{P}(M)$ je množica skupin srečanj. Vsaka skupina srečanj vsebuje vsaj eno srečanje in vsako srečanje je v natanko eni skupini srečanj. Za poljubni srečanji $m_i = (l_i, s_i, g_i, n_i, h_i, k_i)$ in $m_j = (l_j, s_j, g_j, n_j, h_j, k_j)$ iz iste skupine srečanj

mora veljati $l_i = l_j$, $n_i = n_j$, $h_i = h_j$, $k_i = k_j$ in $g_i \neq g_j$.



Slika 2: Srečanja in skupine srečanj.

16. Množica $P \subseteq U \times \mathbb{N}^2$ je množica vseh predavanj, kjer predavanje $p = (u, v, z)$ pomeni, da je predavanje v-ti del skupine srečanj u in traja z ur. Naj ima skupina srečanj u periodo n tednov, trajanje h ur in večkratnost k. Potem mora biti v eni periodi urnika vseh predavanj $p_i = (u, i, z_i)$, ki vsebujejo skupino srečanj u, natanko $\frac{n_P}{n}\, k$ in skupaj morajo trajati $\frac{n_P}{n}\, h$ ur. Veljati mora torej

$$\sum_{i=1}^{\frac{n_P}{n}\, k} z_i = \frac{n_P}{n}\, h.$$

17. Množica $W \subseteq P \times R \times T$ je množica fiksnih razvrstitev predavanj. Fiksna razvrstitev $w = (p, r, t)$ pomeni, da je predavanju p dodeljena predavalnica r in ura začetka t.

Ker je definicija problema urnika zapletena, bomo za lažje razumevanje posamezne elemente iz definicije razložili na primerih:

Skupine študentov. Vsak razred razdelimo na eno ali več skupin študentov. Predavanja ene skupine se nikoli ne smejo prekrivati.

Če imajo vsi študenti nekega razreda enak predmetnik, potem skupine študentov v definiciji problema urnika ustrezajo skupinam študentov, na katere je razred razdeljen v realnosti. V tem primeru razred razdelimo na več skupin le, če se kak predmet izvaja po skupinah. Recimo, da se predmet s_1 izvaja hkrati za vse študente nekega razreda, predmet s_2 pa se izvaja dvakrat – vsakič za en del študentov. V problemu urnika razred

razdelimo na dve skupini, ki obiskujeta predmet s_2 neodvisno ena od druge, predmet s_1 pa obiskujeta hkrati (to dosežemo s pomočjo skupin srečanj, glej spodaj).

Navedimo še en primer. Recimo, da so študenti pri predmetu s_1 razdeljeni v 2 skupini, pri predmetu s_2 v 3 skupine, pri ostalih predmetih pa predavanja obiskujejo vsi skupaj. Pri definiranju problema urnika naredimo 6 skupin $g_1, \ldots, g_6$. Možna razporeditev predavanj je naslednja: predmet s_1 enkrat poslušajo skupaj skupine g_1, g_2 in g_3, drugič pa skupine g_4, g_5 in g_6. Podobno predmet s_2 obiskujeta posebej skupini g_1 in g_2, posebej skupini g_3 in g_4 ter posebej skupini g_5 in g_6. Ostale predmete poslušajo vse skupine hkrati.

Če so v nekem razredu možni izbirni predmeti, skupine niso več „prave" skupine študentov, saj vsi študenti ene skupine ne obiskujejo vseh predavanj te skupine. S skupinami takrat določimo, kako se predmeti lahko prekrivajo. Če npr. želimo, da se vsa predavanja nekega razreda nikoli ne prekrivajo, iz razreda naredimo le eno skupino. Ta način določanja skupin predvideva, da vemo, koliko študentov bo obiskovalo nek razred, ne vemo pa, kako bodo izbirali izbirne predmete.

Študentom, ki so hkrati vpisani na dva študijska programa (vzporedni študij), se predavanja lahko prekrivajo, saj jih pri sestavljanju urnika ne upoštevamo.

Predmeti in predavalnice. Nekateri predmeti, kot so npr. računalništvo ali laboratorijske vaje, potrebujejo točno določene predavalnice (računalniško predavalnico ali laboratorij). Pravimo, da ima predmet *zahtevane* predavalnice. Vsaki predavalnici moramo določiti ekskluzivnost. Ekskluzivne predavalnice so navadno posebne predavalnice (npr. laboratorij), za katere želimo, da se v njih izvajajo samo tisti predmeti, ki te predavalnice zahtevajo. Če imamo problem urnika, v katerem predmet računalništvo zahteva računalniško učilnico, predmet laboratorijske vaje pa laboratorij, ki je ekskluzivna predavalnica, potem se v laboratoriju lahko izvajajo le laboratorijske vaje, v računalniški učilnici pa računalništvo in tudi drugi predmeti.

Srečanja, skupine srečanj in predavanja. Recimo, da imamo problem urnika s periodo 2 tedna. Dve skupini g_1 in g_2 istega razreda morata skupaj poslušati predmet s s trajanjem 1.5 ure na teden. Želimo, da predmet predava predavatelj l enkrat na 14 dni. Torej imamo dve srečanji m_1 in m_2 z naslednjimi podatki: $m_1 = (l, s, g_1, 2, 1, 3)$ in $m_2 = (l, s, g_2, 2, 1, 3)$. Ker želimo, da se predmet predava hkrati za obe skupini, srečanji združimo v eno skupino srečanj $u_1 = \{m_1, m_2\}$. Večkratnost srečanj je enaka 1, zato iz skupine srečanj dobimo le eno predavanje s podatki $p = (u_1, 1, 3)$.

Če imamo v istem problemu urnika skupino srečanj u s trajanjem 5 ur, periodo 1 teden in večkratnostjo 2 (trajanje skupine srečanj se razdeli na 3 + 2 uri), potem se skupina srečanj razdeli na naslednja predavanja: $p_1 = (u, 1, 3)$, $p_2 = (u, 2, 2)$, $p_3 = (u, 3, 3)$ in $p_4 = (u, 4, 2)$. Predavanji p_1 in p_3 bosta potekali ob enakem času vsako v svojem tednu. (Enako velja za predavanji p_2 in p_4.)

Prepovedane ure. Z določanjem prepovedanih ur za predavatelje in predavalnice lahko dosežemo tudi to, da se nobeno predavanje ne odvija v soboto in nedeljo. Dovolj je, da vsem predavateljem prepovemo vse sobotne in nedeljske ure.

Podobno kot smo definirali problem urnika, lahko definiramo tudi njegovo rešitev.

Definicija 2.2. *Rešitev problema urnika* $\mathcal{T}$ je množica razvrstitev

$$X \subseteq P \times R \times T,$$

ki iz predavanj sestavi urnik. Razvrstitev $x = (p, r, t)$ predavanju $p = (u, v, z) \in P$ priredi predavalnico $r \in R$ od začetne ure $t \in T$ do končne ure $t + z \in T$. Za vsako predavanje mora obstajati natanko ena razvrstitev. Dopustna rešitev urnika mora zadostiti naslednjim pogojem (strogim omejitvam):

1. Upoštevati moramo fiksne razvrstitve W. Če je $w = (p, r_w, t_w)$ fiksna razvrstitev predavanja p, mora za njegovo razvrstitev $x = (p, r_x, t_x)$ veljati:

   $$r_x = r_w \text{ in } t_x = t_w.$$

2. Skupina srečanj u s periodo n in večkratnostjo k je razdeljena na natanko $\frac{n_P}{n}\, k$ predavanj. Če je perioda srečanja n manjša od periode urnika n_P, se morajo razvrstitve po k predavanj ponoviti vsak n-ti teden.
   Drugače povedano: za $i = 1, \ldots, k$ in $j = 0, \ldots, \frac{n_P}{n} - 1$ so

   $$p_{i+kj} = (u, i + kj, z_i)$$

   vsa predavanja skupine srečanj u. Za razvrstitve teh predavanj mora veljati:

   $$x_{i+kj} = (p_{i+kj}, r_i, t_i + 7j\, n_D\, n),$$

   kjer je $1 \leq t_i \leq 7\, n_D\, n$.

3. Predavatelj ne sme imeti več predavanj hkrati. Če sta $p_1 = (u_1, v_1, z_1)$ in $p_2 = (u_2, v_2, z_2)$ dve predavanji istega predavatelja, potem razvrstitvi $x_1 = (p_1, r_1, t_1)$ in $x_2 = (p_2, r_2, t_2)$ ne smeta potekati istočasno. Veljati mora bodisi $t_1 + z_1 \leq t_2$ bodisi $t_2 + z_2 \leq t_1$.

4. Predavateljem ob nekaterih urah ne smemo dodeliti predavanj. Če so predavanja $p_i = (u_i, v_i, z_i)$, $i = 1, \ldots, d$, vsa predavanja predavatelja l, potem mora za vse razvrstitve $x_i = (p_i, r_i, t_i)$ teh predavanj veljati

$$(l, t_j) \notin A \text{ za vse } t_j \in \mathbb{N}, \text{ za katere je } t_i \leq t_j < t_i + z_i \text{ za kak } i \in \{1, \ldots, d\}.$$

5. Tudi skupina ne sme imeti več predavanj hkrati. Če sta $p_1 = (u_1, v_1, z_1)$ in $p_2 = (u_2, v_2, z_2)$ dve predavanji iste skupine študentov, potem razvrstitvi $x_1 = (p_1, r_1, t_1)$ in $x_2 = (p_2, r_2, t_2)$ ne smeta potekati istočasno. Veljati mora bodisi $t_1 + z_1 \leq t_2$ bodisi $t_2 + z_2 \leq t_1$.

6. Predavalnica ne sme biti dodeljena več predavanjem hkrati. Naj bosta $p_1 = (u_1, v_1, z_1)$ in $p_2 = (u_2, v_2, z_2)$ dve predavanji, katerih razvrstitvi $x_1 = (p_1, r, t_1)$ in $x_2 = (p_2, r, t_2)$ potekata v isti predavalnici r. Potem predavanji ne smeta potekati istočasno. Veljati mora bodisi $t_1 + z_1 \leq t_2$ bodisi $t_2 + z_2 \leq t_1$.

7. Vsa predavanja se morajo odvijati v dovolj velikih predavalnicah. Naj bo predavanje $p = (u, v, z)$ del skupine srečanj $u = \{m_1, \ldots, m_{|u|}\}$, kjer je $m_i = (l, s_i, g_i, n, h, k)$ za vsak i. Potem mora za razvrstitev $x = (p, r, t)$ predavanja p veljati

$$\sum_i |g_i| \leq |r|.$$

8. Upoštevati moramo, da so za nekatera predavanja potrebne posebne predavalnice. Če se na predavanju p predavajo predmeti $s_1, \ldots, s_d$, od katerih vsaj eden zahteva posebne učilnice ($\eta(s_i) \neq \{\}$ za vsaj en i), potem mora za razvrstitev $x = (p, r, t)$ predavanja p veljati

$$r \in \bigcap_{\substack{i=1 \\ \eta(s_i) \neq \{\}}}^{d} \eta(s_i).$$

9. Ekskluzivnim predavalnicam lahko dodelimo le predavanja, ki zahtevajo te predavalnice. Naj se na predavanju p predavajo predmeti $s_1, \ldots, s_d$. Če je $x = (p, r, t)$

razvrstitev predavanja p, ki vsebuje ekskluzivno predavalnico ($\nu(r) = 1$), morajo vsi predmeti s_i zahtevati predavalnico r:

$$r \in \eta(s_i) \text{ za vsak } i \in \{1, \ldots, d\}.$$

10. Predavalnice so lahko ob nekaterih terminih rezervirane, zato jim takrat ne smemo dodeliti predavanj. Za razvrstitev $x = (p, r, t)$ predavanja $p = (u, v, z)$, ki poteka v predavalnici r, mora veljati

$$(r, t_j) \notin B \text{ za vse } t_j \in \mathbb{N}, \text{ za katere je } t \leq t_j \leq t + z.$$

11. Razvrstitev $x = (p, r, t)$ nekega predavanja $p = (u, v, z)$, ki traja več kot eno uro ($z > 1$), se mora zgoditi v enem dnevu. Če je n_D število ur v dnevu, mora biti

$$\left\lfloor \frac{t-1}{n_D} \right\rfloor = \left\lfloor \frac{t+z-1}{n_D} \right\rfloor.$$

12. Predavanja, ki nastanejo iz iste skupine srečanj, se lahko odvijajo le v različnih dnevih. Če je u skupina srečanj z večkratnostjo $k > 1$, potem mora za razvrstitve $x_i = (p_i, r_i, t_i)$ vseh predavanj $p_i = (u, i, z_i)$, ki so i-ti del skupine srečanj u, veljati, da je

$$\left\lfloor \frac{t_i - 1}{n_D} \right\rfloor \neq \left\lfloor \frac{t_j - 1}{n_D} \right\rfloor \text{ za } i \neq j, \quad i, j = 1, \ldots, k.$$

Zdaj imamo definirano tudi rešitev problema urnika. Pokažimo na primeru, kako lahko iz rešitve kot množice razvrstitev dobimo tak urnik, kot si ga sicer predstavljamo. Ogledali si bomo dva primera. Prvi je urnik za predavalnico (podobno izgleda tudi urnik za predavatelja), kjer se ure ne prekrivajo in je zato urnik bolj enostaven. Drugi pa je urnik za razred, v katerem so študenti razdeljeni na skupine in se zato nekatera predavanja razreda prekrivajo.

Primer 2.1. Rešitev problema urnika $\mathcal{T}$ (perioda je dolga en teden, ure gredo od 7-ih do 19-ih) je množica razvrstitev $X \subseteq P \times R \times T$. Zanima nas urnik predavalnice r, ki ga dobimo tako, da pregledamo vse razvrstitve s predavalnico r. Razvrstitve uredimo glede na uro:

$$\begin{array}{llll} x_1 = (p_1, r, 3) & x_2 = (p_2, r, 5) & x_3 = (p_3, r, 14) & x_4 = (p_4, r, 15) \\ x_5 = (p_5, r, 27) & x_6 = (p_6, r, 30) & x_7 = (p_7, r, 40) & x_8 = (p_8, r, 54) \end{array}$$

Zapišimo še vsa predavanja, ki nastopajo v razvrstitvah. Tako lahko vidimo, koliko ur trajajo predavanja:

$$p_1 = (u_1, 1, 2) \quad p_2 = (u_2, 2, 3) \quad p_3 = (u_3, 1, 1) \quad p_4 = (u_1, 2, 2)$$
$$p_5 = (u_4, 1, 2) \quad p_6 = (u_5, 1, 3) \quad p_7 = (u_2, 1, 2) \quad p_8 = (u_6, 1, 1)$$

Urnik predavalnice r predstavimo s tabelo 2, v kateri označimo ure, ko je predavalnica zasedena s skupino srečanj, ki predavalnico uporablja.

	pon	tor	sre	čet	pet	sob	ned
7^h	1	13	25	37	49	61	73
8^h	2	14 u_3	26	38	50	62	74
9^h	3 u_1	15 u_1	27 u_4	39	51	63	75
10^h	4 u_1	16 u_1	28 u_4	40 u_2	52	64	76
11^h	5 u_2	17	29	41 u_2	53	65	77
12^h	6 u_2	18	30 u_5	42	54 u_6	66	78
13^h	7 u_2	19	31 u_5	43	55	67	79
14^h	8	20	32 u_5	44	56	68	80
15^h	9	21	33	45	57	69	81
16^h	10	22	34	46	58	70	82
17^h	11	23	35	47	59	71	83
18^h	12	24	36	48	60	72	84

Tabela 2: Urnik za predavalnico r.

△

Primer 2.2. Spet imamo dano rešitev problema urnika $\mathcal{T}$ (perioda je dolga en teden, ure gredo od 7-ih do 19-ih) kot množico razvrstitev $X \subseteq P \times R \times T$. Tokrat nas zanima urnik razreda c, ki je razdeljen na skupine g_1, g_2 in g_3. Označimo z m^i vsa srečanja, v katerih nastopa skupina g_i:

$$m_1^1 = (l_1, s_1, g_1, 1, 4, 2) \quad m_1^2 = (l_1, s_1, g_2, 1, 4, 2) \quad m_1^3 = (l_1, s_1, g_3, 1, 4, 2)$$
$$m_2^1 = (l_1, s_2, g_1, 1, 6, 2) \quad m_2^2 = (l_2, s_{11}, g_2, 1, 2, 1) \quad m_2^3 = (l_1, s_2, g_3, 1, 6, 2)$$
$$m_3^1 = (l_2, s_3, g_1, 1, 3, 1) \quad m_3^2 = (l_2, s_3, g_2, 1, 3, 1) \quad m_3^3 = (l_3, s_4, g_3, 1, 2, 1)$$
$$m_4^1 = (l_3, s_4, g_1, 1, 2, 1) \quad m_4^2 = (l_5, s_6, g_2, 1, 4, 2) \quad m_4^3 = (l_7, s_8, g_3, 1, 2, 1)$$
$$m_5^1 = (l_4, s_5, g_1, 1, 3, 2) \quad m_5^2 = (l_7, s_8, g_2, 1, 2, 1) \quad m_5^3 = (l_4, s_7, g_3, 1, 3, 2)$$
$$m_6^1 = (l_5, s_6, g_1, 1, 4, 2) \quad m_6^2 = (l_3, s_9, g_2, 1, 6, 2) \quad m_6^3 = (l_3, s_{10}, g_3, 1, 4, 2)$$
$$m_7^1 = (l_4, s_7, g_1, 1, 3, 1) \quad m_7^2 = (l_8, s_{11}, g_2, 1, 2, 1) \quad m_7^3 = (l_9, s_{12}, g_3, 1, 1, 1)$$
$$m_8^1 = (l_6, s_8, g_1, 1, 2, 1) \quad m_8^3 = (l_6, s_{13}, g_3, 1, 3, 1)$$

V srečanju $m_5^1 = (l_4, s_5, g_1, 1, 3, 2)$ predava predavatelj l_4 skupini g_1 predmet s_5. Srečanje bo moralo imeti v periodi, dolgi en teden, tri ure, razdeljene na dve predavanji.

Izpišimo skupine srečanj, ki vsebujejo zgoraj našteta srečanja:

u_1=$\{m_1^1, m_1^2, m_1^3\}$	u_5=$\{m_5^1, m_5^3\}$	u_9=$\{m_2^2\}$	u_{13}=$\{m_6^3\}$
u_2=$\{m_2^1, m_2^3\}$	u_6=$\{m_6^1, m_4^2\}$	u_{10}=$\{m_5^2, m_4^3\}$	u_{14}=$\{m_7^3\}$
u_3=$\{m_3^1, m_3^2\}$	u_7=$\{m_7^1\}$	u_{11}=$\{m_6^2\}$	u_{15}=$\{m_8^3\}$
u_4=$\{m_4^1, m_3^3\}$	u_8=$\{m_8^1\}$	u_{12}=$\{m_7^2\}$	

Kot lahko vidimo, so v eni skupini srečanj samo srečanja, ki se razlikujejo le v skupini študentov in predmetu. Ostale komponente srečanj iste skupine srečanj se morajo ujemati.

Skupine srečanj se razdelijo na toliko predavanj, kolikor je večkratnost srečanja. Napišimo vsa predavanja za navedene skupine srečanj:

p_1=$(u_1, 1, 2)$	p_8=$(u_5, 2, 1)$	p_{15}=$(u_{11}, 1, 3)$
p_2=$(u_1, 2, 2)$	p_9=$(u_6, 1, 2)$	p_{16}=$(u_{11}, 2, 3)$
p_3=$(u_2, 1, 3)$	p_{10}=$(u_6, 2, 2)$	p_{17}=$(u_{12}, 1, 2)$
p_4=$(u_2, 2, 3)$	p_{11}=$(u_7, 1, 3)$	p_{18}=$(u_{13}, 1, 2)$
p_5=$(u_3, 1, 3)$	p_{12}=$(u_8, 1, 2)$	p_{19}=$(u_{13}, 2, 2)$
p_6=$(u_4, 1, 2)$	p_{13}=$(u_9, 1, 2)$	p_{20}=$(u_{14}, 1, 1)$
p_7=$(u_5, 1, 2)$	p_{14}=$(u_{10}, 1, 2)$	p_{21}=$(u_{15}, 1, 3)$

Za lažje razumevanje napisanega si podrobneje oglejmo skupino srečanj $u_5 = \{m_5^1, m_5^3\}$. Skupina ima 3 ure in večkratnost 2 (podatka sta zapisana pri srečanjih m_5^1 in m_5^3). To pomeni, da bo skupina srečanj u_5 razdeljena na dve predavanji. Prvo bo dolgo 2 uri ($p_7 = (u_5, 1, 2)$), drugo pa 1 uro ($p_8 = (u_5, 2, 1)$).

Naštejmo še vse razvrstitve za dana predavanja in jih uredimo glede na uro:

x_1=$(p_1, r_1, 2)$	x_8=$(p_8, r_2, 15)$	x_{15}=$(p_{15}, r_2, 38)$
x_2=$(p_2, r_2, 2)$	x_9=$(p_9, r_1, 17)$	x_{16}=$(p_{16}, r_1, 39)$
x_3=$(p_3, r_1, 5)$	x_{10}=$(p_{10}, r_1, 19)$	x_{17}=$(p_{17}, r_2, 40)$
x_4=$(p_4, r_2, 5)$	x_{11}=$(p_{11}, r_2, 19)$	x_{18}=$(p_{18}, r_2, 43)$
x_5=$(p_5, r_1, 6)$	x_{12}=$(p_{12}, r_1, 26)$	x_{19}=$(p_{19}, r_3, 50)$
x_6=$(p_6, r_1, 8)$	x_{13}=$(p_{13}, r_3, 29)$	x_{20}=$(p_{20}, r_1, 52)$
x_7=$(p_7, r_1, 14)$	x_{14}=$(p_{14}, r_1, 31)$	x_{21}=$(p_{21}, r_1, 54)$

Razvrstitve zapišemo v tabeli, ki bo predstavljala urnik razreda c (tabela 3). Za vsako uro na urniku (izpustili smo ure, ki so brez predavanj) navedemo predavatelja in predavalnico, v kateri bo predavanje potekalo. Nato naštejemo še vse predmete in skupine študentov, ki se bodo skupaj srečali v tej predavalnici. Urnik, ki ga tako dobimo, je zapleten zaradi prekrivanja ur razreda.

	pon		tor		sre	čet		pet
7^h	1		13		25	37		49
8^h	$r_1\ l_1$ $s_2\ g_1\ g_3$	$r_2\ l_3$ $s_9\ g_2$	$r_1\ l_4$ $s_7\ g_1$		$r_1\ l_2$ $s_3\ g_1\ g_2$	$r_2\ l_3$ $s_4\ g_1\ g_3$		$r_3\ l_1$ $s_1\ g_1\ g_2\ g_3$
9^h	$r_1\ l_1$ $s_2\ g_1\ g_3$	$r_2\ l_3$ $s_9\ g_2$	$r_1\ l_4$ $s_7\ g_1$	$r_2\ l_8$ $s_{11}\ g_2$	$r_1\ l_2$ $s_3\ g_1\ g_2$	$r_2\ l_3$ $s_4\ g_1\ g_3$	$r_1\ l_3$ $s_9\ g_2$	$r_3\ l_1$ $s_1\ g_1\ g_2\ g_3$
10^h	$r_1\ l_1$ $s_2\ g_1\ g_3$	$r_2\ l_3$ $s_9\ g_2$	$r_1\ l_4$ $s_7\ g_1$	$r_2\ l_8$ $s_{11}\ g_2$	$r_1\ l_2$ $s_3\ g_1\ g_2$	$r_2\ l_1$ $s_2\ g_1\ g_3$	$r_1\ l_3$ $s_9\ g_2$	$r_1\ l_4$ $s_5\ g_1$ $s_7\ g_3$
11^h	$r_1\ l_4$ $s_5\ g_1$ $s_7\ g_3$	$r_2\ l_2$ $s_{11}\ g_2$	$r_1\ l_5$ $s_6\ g_1\ g_2$		$r_3\ l_1$ $s_1\ g_1\ g_2\ g_3$	$r_2\ l_1$ $s_2\ g_1\ g_3$	$r_1\ l_3$ $s_9\ g_2$	$r_1\ l_4$ $s_5\ g_1$ $s_7\ g_3$
12^h	$r_1\ l_3$ $s_{10}\ g_3$	$r_2\ l_2$ $s_{11}\ g_2$	$r_1\ l_5$ $s_6\ g_1\ g_2$		$r_3\ l_1$ $s_1\ g_1\ g_2\ g_3$	$r_2\ l_1$ $s_2\ g_1\ g_3$		$r_1\ l_6$ $s_{13}\ g_3$
13^h	$r_1\ l_3$ $s_{10}\ g_3$		$r_1\ l_6$ $s_8\ g_1$	$r_2\ l_7$ $s_8\ g_2\ g_3$	$r_1\ l_5$ $s_6\ g_1\ g_2$	$r_2\ l_2$ $s_{10}\ g_3$		$r_1\ l_6$ $s_{13}\ g_3$
14^h	$r_1\ l_9$ $s_{12}\ g_3$		$r_1\ l_6$ $s_8\ g_1$	$r_2\ l_7$ $s_8\ g_2\ g_3$	$r_1\ l_5$ $s_6\ g_1\ g_2$	$r_2\ l_2$ $s_{10}\ g_3$		$r_1\ l_6$ $s_{13}\ g_3$

Tabela 3: Urnik za razred s skupinami g_1, g_2 in g_3.

△

V definiciji 2.2 smo opisali vse dopustne rešitve problema urnika. Izmed vseh iščemo tisto, ki bo najboljša. Merilo kakovosti urnika nam bo dala funkcija, ki jo bomo sestavili iz uteženih šibkih omejitev. Problem urnika je torej optimizacijski problem.

Definicija 2.3. *Optimizacijska naloga* je naloga oblike:

Iščemo max (ali min) funkcije $f(x)$ za $x \in \mathcal{D}$.

Funkcija $f\colon\ \mathcal{D} \to \mathbb{R}$ se imenuje *kriterijska funkcija*, množica $\mathcal{D}$ pa *množica dopustnih rešitev*. Točke $x \in \mathcal{D}$, kjer je dosežen max (ali min), so *rešitve optimizacijske naloge*.

Pri problemu urnika množico dopustnih rešitev $\mathcal{D}$ sestavljajo dopustni urniki (to so vsi urniki, ki zadoščajo strogim omejitvam), kriterijska funkcija pa je funkcija uteženih šibkih omejitev.

V našem primeru bo kriterijska funkcija sestavljena iz naslednjih omejitev:

1. *Predavanja naj se izvajajo ob urah, ki so najugodnejše za poučevanje.*
   Primernost ur za poučevanje predstavimo s tabelo, v kateri za vsak razred c vsaki uri t_i dodelimo utež α_i. Večja kot je utež, bolj je ura zaželena. Za naš primer privzemimo, da je perioda dolga en teden in da se delovni dan začne ob sedmih zjutraj ter traja 12 ur. V tabeli 4 prikazujemo primer obtežitve ur za razred c (števila levo zgoraj so ure, desno pa uteži).

	pon	tor	sre	čet	pet	sob	ned
7^h	1 1	13 1	25 1	37 1	49 1	61 0	73 0
8^h	2 4	14 10	26 10	38 10	50 10	62 0	74 0
9^h	3 10	15 10	27 10	39 10	51 10	63 0	75 0
10^h	4 10	16 10	28 10	40 10	52 10	64 0	76 0
11^h	5 10	17 10	29 10	41 10	53 10	65 0	77 0
12^h	6 10	18 10	30 10	42 10	54 10	66 0	78 0
13^h	7 10	19 10	31 10	43 10	55 4	67 0	79 0
14^h	8 10	20 10	32 10	44 4	56 2	68 0	80 0
15^h	9 4	21 4	33 4	45 2	57 1	69 0	81 0
16^h	10 2	22 2	34 2	46 1	58 0	70 0	82 0
17^h	11 1	23 1	35 1	47 1	59 0	71 0	83 0
18^h	12 1	24 1	36 1	48 1	60 0	72 0	84 0

Tabela 4: Obtežitev ur za razred c.

Šibko omejitev zapišemo s funkcijo. Naj bo $x_i = (p_i, r_i, t_i)$ razvrstitev predavanja $p_i = (u_i, v_i, z_i)$, pri katerem skupino srečanj zapišemo kot $u_i = \{m_{i,1}, \ldots, m_{i,|u_i|}\}$. Ker je obtežitev ur odvisna od razreda, označimo z $\alpha_{i,j}$ utež ure t_i srečanja $m_{i,j}$. Za vsako razvrstitev x_i lahko zapišemo vsoto uteži ur $t_i, t_i + 1, \ldots, t_i + z_i - 1$ kot

$$\sum_{j=1}^{|u_i|} (\alpha_{i,j} + \cdots + \alpha_{i+z_i-1,j}) .$$

Funkcija

$$f_1(X) = \sum_{x_i \in X} \sum_{j=1}^{|u_i|} (\alpha_{i,j} + \cdots + \alpha_{i+z_i-1,j})$$

nam vrne vsoto uteži vseh ur za vse razvrstitve.

2. *Urnik naj vsebuje čim manj prostih ur za študente.*
   Za vsako skupino študentov $g \in G$ naj bo $P_g = \{p_i = (u_i, v_i, z_i)\}_{i \in I_g}$ množica predavanj, ki vsebujejo skupino g, in $X_g = \{x_i = (p_i, r_i, t_i)\}_{i \in I_g}$ urejena množica razvrstitev predavanj iz P_g. Za vsako skupino g je množica X_g urejena naraščajoče po urah t_i. To pomeni, da za vsak $i = 2, \ldots, |I_g|$ velja $t_i > t_{i-1}$. Radi bi sešteli vse proste ure med predavanji skupine g. Za dve zaporedni predavanji p_{i-1} in p_i, ki potekata v istem dnevu, torej $\left\lfloor \frac{t_{i-1}-1}{n_D} \right\rfloor = \left\lfloor \frac{t_i-1}{n_D} \right\rfloor$, izračunamo število prostih ur med njima in ga označimo s ξ_i:

   $$\xi_i = t_i - (t_{i-1} + z_{i-1}).$$

   Če seštejemo vse proste ure vseh skupin študentov, dobimo število prostih ur v razvrstitvi X. Ker želimo, da je prostih ur čim manj, vzamemo za funkcijo $f_2(X)$ npr. recipročno vrednost vsote vseh prostih ur:

   $$f_2(X) = \left(1 + \sum_{g \in G} \sum_{i=2}^{|I_g|} \xi_i \right)^{-1}.$$

   K vsoti vseh prostih ur smo 1 prišteli zato, da nikoli ne delimo z 0.

3. *Študenti naj se čim manj selijo iz ene stavbe v drugo in iz ene predavalnice v drugo.*
   Naj bosta za vsako skupino študentov $g \in G$ množici P_g in X_g definirani kot v prejšnji točki. Za ure, ki si v enem dnevu sledijo, torej za $i = 1, \ldots, |I_g| - 1$, za katere je $t_{i+1} = t_i + z_i - 1$ in $\left\lfloor \frac{t_{i+1}-1}{n_D} \right\rfloor = \left\lfloor \frac{t_i-1}{n_D} \right\rfloor$, definiramo funkcijo χ_i, ki bo predstavljala kazen za selitev skupine študentov. Funkcija χ_i bo enaka 0 za vsak par razvrstitev, pri katerem študenti skupine g ostanejo v isti predavalnici, in enaka β_1 za vsak par razvrstitev, pri katerem študenti zamenjajo predavalnico, a ostanejo v isti stavbi. Največjo kazen β_2 dodelimo v primeru, da sta predavalnici v različnih stavbah:

   $$\chi_i = \begin{cases} 0, & \text{če je } r_i = r_{i+1} \\ \beta_1, & \text{če je } r_i \neq r_{i+1} \text{ in sta predavalnici v isti stavbi} \\ \beta_2, & \text{če sta predavalnici } r_i \text{ in } r_{i+1} \text{ v različnih stavbah} \end{cases}$$

Kazni $0 \leq \beta_1 \leq \beta_2$ sta realni števili. Iz funkcij χ_i nato sestavimo funkcijo

$$f_3(X) = \left(1 + \sum_{g \in G} \sum_{i \in I_g} \chi_i\right)^{-1},$$

ki nam vrne recipročno vrednost vsote vseh kazni pri selitvah študentov med odmori. Funkcija $f_3(X)$ je tem večja, čim manjše so kazni za selitve. (Podobno kot v prejšnji točki smo tudi tu 1 dodali zato, da nikoli ne delimo z 0.)

Iz opisanih šibkih omejitev lahko sedaj sestavimo kriterijsko funkcijo. Naj bodo števila $\omega_1, \omega_2, \omega_3 \in \mathbb{R}^+$ uteži. Funkcija

$$f(X) = \omega_1 f_1(X) + \omega_2 f_2(X) + \omega_3 f_3(X)$$

je kriterijska funkcija za naš problem urnika. Razvrstitev X^*, za katero velja

$$f(X^*) = \max_X f(X),$$

je optimalna rešitev problema urnika.

2.3 Metode reševanja

V 60-ih letih so za pomoč pri reševanju problema urnika začeli uporabljati računalnike. Hitro so videli, da je problem zahteven. Leta 1976 so Even, Itai in Shamir v [19] pokazali, da je problem urnika NP-težek problem.

Definicija 2.4. *Problem* je *NP-težek*, če lahko vsak problem iz razreda NP prevedemo nanj v polinomskem času.

V dokazu so Even, Itai in Shamir na problem urnika prevedli problem izpolnljivosti (3 SAT). (Povzetek tega dokaza je opisan tudi v [23].)

Za NP-težke probleme ne poznamo točnih učinkovitih algoritmov in verjetno ne obstajajo, zato v praksi za take probleme iščemo „zasilne izhode”:

- *Aproksimacijski algoritmi* najdejo rešitev, ki je dokazljivo „blizu” optimalne.
- *Verjetnostni algoritmi* uporabljajo generator slučajnih števil. Rezultat, ki ga dobijo, je pravilen z določeno verjetnostjo.
- *Lokalna optimizacija* je postopek, pri katerem najdemo rešitev, ki je lokalni optimum v prostoru dopustnih rešitev.

- *Hevristike* so metode, ki bi „po zdravi pameti" morale pripeljati do dobre rešitve.

Za reševanje problema urnika se uporabljajo različne metode [2, 10, 16, 18, 20], ki so zbrane v tabeli 5. V nadaljevanju si bomo te metode podrobneje ogledali.

Točne metode	**Hevristike**	**Metahevristike**	**Interaktivne metode**
sestopanje	požrešni algoritmi	tabu iskanje	popravljanje baze
razveji in omeji	nepopolno	simulirano	podatkov
dinamično	sestopanje	ohlajanje	pregledovanje ročno
programiranje	dekompozicija	logično	narejenih urnikov
		programiranje	posebni urejevalniki
	pravila odločanja	z omejitvami	urnikov
	funkcije prednosti	genetski algoritmi	
Kombinirane metode			
sistemi za podporo odločanju			

Tabela 5: Klasifikacija metod za reševanje problema urnika.

2.3.1 Točne metode

Raziskovalci so v preteklosti za probleme podobne optimizaciji uporabljali splošne metode za reševanje problemov, kot so **sestopanje**, **razveji in omeji**, **dinamično programiranje** in druge. S časom pa so se točne metode izkazale za premalo uporabne pri reševanju resničnih problemov razporejanja. Le probleme, kjer opustimo večino zahtev, lahko učinkovito rešujemo s točnimi metodami. Zato je večina programov, ki rešuje problem urnika, narejena na osnovi hevurističnih metod.

2.3.2 Hevristike

Prve hevristične metode za reševanje problema urnika so simulirale ročno sestavljanje urnika in so temeljile na „zdravi pameti". Najpogosteje se uporabljajo naslednje hevristike:

Požrešni algoritmi. Požrešni algoritmi izberejo posamezno razvrstitev z nekim lokalno optimizacijskim pravilom in ne spreminjajo predhodnih razvrstitev. Taki algoritmi ne znajo priti iz slepe ulice, ko ne morejo razvrstiti nobenega predavanja več.

Nepopolno sestopanje. Nepopolno sestopanje izbere posamezno razvrstitev z nekim lokalno optimizacijskim pravilom, a lahko spremeni tudi predhodne razvrstitve, če za trenutno predavanje ne obstaja dopustna razvrstitev.

Dekompozicija. Dekompozicija je redukcija danega problema na podobne, a manjše probleme, ki jih, če se le da, spet reduciramo. Najpogosteje uporabljena dekompozicija je redukcija problema urnika glede na časovne periode, kot so tedni ali dnevi.

Naštete hevristike so bile s številnimi modifikacijami in kombinacijami uporabljene v mnogih programih, ki rešujejo problem urnika. Hevristike se med seboj ločijo tudi po tem, kako izberejo smer lokalnega iskanja. Najenostavnejša pravila, ki določajo naslednji korak v lokalnem iskanju, izberejo prvi prosti termin ali naključni prosti termin. Bolj pogosta so **pravila odločanja** in pravila, ki optimizirajo neko **funkcijo prednosti**.

2.3.3 Metahevristike

Glavna težava pri problemu urnika je obstoj številnih dopustnih rešitev, ki so lahko zelo različne ena od druge, in številnih lokalnih ekstremov. Zgoraj opisane hevristike navadno ne zmorejo zapustiti lokalnega ekstrema, da bi našle boljšo rešitev. V zadnjih letih se pri problemu urnika uspešno uporabljajo metahevristike. To so metode, ki težijo k iskanju boljših rešitev od že najdenih lokalnih ekstremov:

Tabu iskanje. Tabu iskanje je lokalno optimizacijska tehnika, ki v procesu dopušča tudi slabše rešitve [13, 29]. To je agresiven postopek, ki hitro pride do lokalnega optimuma z izbiranjem najboljše rešitve v soseščini pri vsaki iteraciji.

Soseščina rešitve x je definirana kot množica rešitev $N(x)$, ki jih iz x lahko dobimo v enem koraku lokalne optimizacije. Osnovna poteza je premik iz rešitve x na najboljšo rešitev $x^* \in N(x)$, tudi če je x^* slabša od x. Ker je v vsaki soseščini dovoljena samo ena poteza, se iskanje oddalji od lokalnih optimumov. Brez izboljšav pa lahko hitro zapade v ciklanje po optimumih. Da se temu izognemo, uporabimo koncept seznama tabujev.

Seznam tabujev T vsebuje opis zadnjih n potez oz. rešitev. Ko pregledujemo soseščino $N(x)$, hkrati pregledamo še seznam tabujev T, da se ne vrnemo na prejšnji korak. Tabu

poteze so prepovedane, razen če rešitev zadošča nekemu kriteriju, npr. da je dobljena rešitev najboljša do tistega trenutka.

Tabu poteze v problemu urnika, kjer je bilo predavanje p premaknjeno z ure t_1 na uro t_2, so lahko ene od naslednjih:

- vsaka poteza, ki bi premaknila predavanje p;
- vsako premikanje z ure t_2;
- vsako premikanje na uro t_1;
- vsako premikanje predavanja p na uro t_1;
- vsako premikanje predavanja p iz ure t_2.

Vsaka izmed naštetih prepovedi bo onemogočila premikanje predavanja p z ure t_2 nazaj na uro t_1, a bo hkrati onemogočila tudi različno skupino potez. Prve tri prepovedi so bolj omejujoče od zadnjih dveh.

Čeprav seznami tabu potez v praksi dovolj dobro zagotavljajo, da se metoda ne zacikla, ne omilijo agresije metode in torej ne zagotovijo, da bi iskanje pokrilo dovolj veliko območje. Tako osnovna implementacija pogosto vključuje razne mehanizme, ki iskanje usmerijo na še neraziskana območja. V najosnovnejši obliki ti mehanizmi prepovejo pogosto uporabljene poteze. Na primer, če je bilo predavanje p dodeljeno uri t v več kot izbranem deležu zadnjih n potez, potem bo ta premik kaznovan. S tem dosežemo, da se iskanje premakne v druga območja.

Algoritem se konča po določenem maksimalnem številu iteracij in vrne najboljšo najdeno rešitev.

Simulirano ohlajanje. Simulirano ohlajanje je različica lokalnega iskanja, ki dovoli izbiro v nekem trenutku slabše rešitve, da se prehitro ne ustavimo v lokalnem optimumu [17]. Ta metoda uporablja analogijo s termodinamiko, saj obravnava dopustne rešitve kot snov (skupek molekul). Ko ima snov visoko temperaturo, molekule lahko preskočijo meje, ko jo ohlajamo, pa zapolnijo prostor z najmanjšim potencialom.

Kot ostala lokalna iskanja tudi simulirano ohlajanje potrebuje definicijo problema v obliki prostora rešitev s strukturo soseščin in cenovno funkcijo, ki vsaki rešitvi priredi število. Postopek se začne z naključno rešitvijo iz prostora rešitev in se nadaljuje tako, da se izbere naključna rešitev iz soseščine prve rešitve. Če je sosednja rešitev boljša od začetne, jo vzamemo za tekočo rešitev. Če pa je slabša od začetne za faktor δ, jo

vzamemo za tekočo rešitev z verjetnostjo $e^{-\delta/T}$, kjer se s koraki algoritma T postopoma zmanjšuje proti 0. Ta postopek se ponavlja, dokler ni T tako majhen, da algoritem ne sprejme nobene nove rešitve. Dobljena rešitev je zelo pogosto dober približek optimalne rešitve.

Zaradi analogije s termodinamiko spremenljivki T pravimo temperatura, postopku, v katerem se T zmanjšuje, pa ohlajanje.

Kakovost rešitve je odvisna od formulacije problema v okviru lokalnega iskanja in algoritma ohlajanja. Če je na voljo dovolj časa, je najboljše dolgo počasno ohlajanje. Začetna temperatura T_0 mora biti dovolj visoka, da je verjetnost sprejema slabše rešitve večja, končna temperatura T_{kon} pa dovolj nizka, da bo postopek konvergiral k optimalni rešitvi.

Logično programiranje z omejitvami. Logično programiranje z omejitvami je oblika logičnega programiranja, v kateri imajo spremenljivke nekatere omejitve [1, 24]. Problemu, ki ga lahko opišemo kot problem programiranja z omejitvami, pravimo *problem izpolnitve*. Problem izpolnitve je sestavljen iz končne množice spremenljivk, v kateri je vsaka spremenljivka povezana s končno domeno in končno množico omejitev. Rešitev problema izpolnitve vsaki spremenljivki priredi vrednost iz njene domene, tako da so vse omejitve upoštevane.

Delni problem izpolnitve pa je problem izpolnitve, pri katerem ima vsaka omejitev utež. Tako lahko ločimo omejitve po pomembnosti (stroge in šibke omejitve). Rešitev delnega problema izpolnitve vsaki spremenljivki priredi vrednost iz njene domene, tako da so upoštevane vse stroge omejitve in čim več šibkih omejitev.

Reševanje problema z logičnim programiranjem z omejitvami lahko razdelimo na dve fazi:

1. Najprej pregledamo omejitve vhodnih spremenljivk. Npr. če imamo omejitev $X > Y$, pri čemer ima spremenljivka X domeno $D_X = \{1, \ldots, 5\}$, spremenljivka Y pa domeno $D_Y = \{2, \ldots, 10\}$, potem domeni spremenimo v $D_X = \{3, 4, 5\}$ in $D_Y = \{2, 3, 4\}$.

2. Nato poiščemo eno izmed spremenljivk, ki še nima dodeljene vrednosti (izbor spremenljivke mora biti dobro premišljen), in ji dodelimo vrednost iz njene domene. Potem spet pregledamo in popravimo omejitve ostalih spremenljivk. Če ima katera od spremenljivk po popravljanju prazno domeno, se moramo vrniti korak nazaj in poiskati kakšno drugo vrednost za izbrano spremenljivko.

Genetski algoritmi. Genetski algoritmi z uporabo populacij, selekcije in genetskih operatorjev, kot sta prekrižanje in mutacija, simulirajo naravno evolucijo. Na tak način množico začetnih rešitev (začetno populacijo) razvijajo v vedno boljše in boljše rešitve. Kar dobijo, je aproksimacija optimalne rešitve.

Pri reševanju našega problema urnika bomo uporabili genetske algoritme, zato si jih bomo podrobneje ogledali v naslednjem poglavju.

2.3.4 Interaktivne metode

K interaktivnim metodam štejemo naslednje:

Popravljanje baze podatkov. Številne hevristike ne zagotavljajo, da bodo naredile urnik, pa čeprav obstaja. To dejstvo in potreba po spreminjanju urnika sredi semestra je botrovalo razvoju interaktivnih orodij, ki omogočajo popravljanje vhodnih podatkov ali že narejenega urnika.

Pregledovanje ročno narejenih urnikov. Kot prva uporaba računalnika za reševanje problema urnika so se pojavili programi, ki so preverili pravilnost ročno narejenih urnikov in vračali diagnostike. Nedopustne urnike so sestavljalci potem morali spet ročno pregledati in popraviti. Popravljen urnik so nato še enkrat preverili s programom. Proces je trajal toliko časa, dokler niso dobili dopustnega urnika.

Posebni urejevalniki urnikov. Takšni urejevalniki omogočajo vnos in popravljanje podatkov in urnikov ter vplivajo na avtomatično generiranje urnika tako, da ponudijo urejanje šibkih omejitev. Ta orodja preverijo pravilnost podatkov, preprečijo vnos nepravilnih ali konfliktnih podatkov in opozarjajo na morebitne konflikte in napake. Struktura podatkov je zelo pomembna za minimiziranje časa, ki je potreben za izdelavo urnikov. Ti sistemi so lahko uporabni tudi za pridobivanje informacij o tem, kako narediti urnik. Včasih takšni urejevalniki nimajo možnosti avtomatičnega generiranja urnikov, ampak se uporabljajo le kot pripomoček za sestavljalca.

2.3.5 Kombinirane metode

Program za sestavljanje urnikov, ki je osnovan na **sistemih za podporo odločanju** (angl. decision support systems), združuje:

– notranje podatkovne strukture in/ali dostop do zunanje baze podatkov;

– grafični uporabniški vmesnik;

– vsaj en algoritem za avtomatično reševanje problema urnika, ki ima za osnovo katero od metahevristik.

Oseba, ki dela s takšnim sistemom, lahko vnaša in spreminja vhodne podatke in parametre. Urnik lahko sestavlja od začetka ali pa spreminja urnike, ki jih je program avtomatično sestavil. Uporabniški vmesnik je pregleden in preprost za uporabo.

3 Reševanje problema urnika z genetskim algoritmom

V tem poglavju bomo najprej opisali splošne lastnosti genetskih algoritmov in si nato podrobneje ogledali genetski algoritem za problem urnika.

3.1 Splošne lastnosti genetskih algoritmov

Genetski algoritmi so močno optimizacijsko orodje, ki se zgleduje po principih evolucije. Pogosto lahko najdejo globalno optimalno rešitev tudi pri zelo zapletenih pogojih, zato jih uporabljajo za reševanje številnih problemov. Med drugim se uporabljajo pri naslednjih vrstah problemov [25]:

Optimizacija. Genetski algoritmi se uporabljajo pri številnih optimizacijskih nalogah, vključno z numerično optimizacijo in kombinatoričnimi optimizacijskimi problemi, kot so razporejanje procesov, problem urnika, problem trgovskega potnika idr.

Avtomatično programiranje. Genetski algoritmi se uporabljajo za razvijanje računalniških programov za določene naloge in za konstruiranje drugih računskih struktur.

Ekologija. Genetski algoritmi se v ekologiji uporabljajo za opisovanje in proučevanje ekoloških pojavov, kot sta simbioza in zajedalstvo.

Seveda pa genetski algoritmi optimalne rešitve ne najdejo vedno. Še več, tudi dolg čas izvajanja algoritma nam ne zagotavlja, da bodo našli dobro rešitev. Tako je odvisno

od danega problema in njegovega formalnega opisa, ali je genetski algoritem primerna metoda za reševanje tega problema.

Vpeljimo nekaj terminologije iz biologije, ki jo bomo uporabljali v nadaljevanju. V kontekstu genetskih algoritmov so ti biološki termini uporabljeni v analogiji z resnično biologijo:

- *Osebek* je osnovna enota v genetskem algoritmu in predstavlja kandidata za rešitev problema, ki ga rešujemo z genetskim algoritmom. Vse lastnosti osebka so zapisane v njegovih *kromosomih*. Osnovne enote kromosomov so *geni*. V genetskih algoritmih ima osebek navadno le en kromosom. Zato bomo v nadaljevanju včasih govorili o osebkih, včasih pa o kromosomih, a bomo v obeh primerih mislili na kandidata za rešitev problema.

- *Populacija* je skupina osebkov, ki obstajajo hkrati. Navadno omenjamo dve populaciji: populacijo staršev in populacijo potomcev. V vsakem koraku genetskega algoritma iz populacije staršev s pomočjo genetskih operatorjev dobimo populacijo potomcev.

- *Generacija* je izraz, s katerim želimo označiti populacijo kot del evolucije. Govorimo npr. o prvi, drugi, i-ti generaciji.

- *Selekcija* je izbiranje posameznih osebkov iz populacije. Ti potem običajno predstavljajo podlago za novo generacijo. Zato je pomembno, da pri selekciji izberemo boljše osebke populacije.

- *Prekrižanje* (angl. crossover) je genetski operator, ki deluje na dveh osebkih populacije (starša) in nam vrne dva nova osebka (potomca).

- *Mutacija* je genetski operator, ki deluje na enem samem osebku tako, da ga (navadno naključno) spremeni.

Genetski algoritem je skupno poimenovanje za vse postopke, ki imajo naslednje lastnosti: populacija osebkov, selekcija glede na kakovost osebka, prekrižanje, ki pridela potomce, in naključna mutacija novih potomcev.

Oglejmo si lastnosti genetskih algoritmov skozi postopek izdelave algoritma. Ko želimo narediti genetski algoritem, moramo najprej izbrati način **zapisa podatkov**. Nato se moramo odločiti za tip **selekcije** in za **genetske operatorje**, ki jih bomo uporabljali v algoritmu. Na koncu moramo določiti še vse **parametre** algoritma. Pri

opisovanju bomo dali poudarek na tiste lastnosti, ki se pogosto pojavljajo tudi v genetskih algoritmih za problem urnika.

Zapis. Ko konstruiramo genetski algoritem, moramo najprej izbrati način, kako bomo zapisali kandidata za rešitev – kromosom. To je najvažnejši faktor, ki pogosto odloča o učinkovitosti izvajanja algoritma. Večina genetskih algoritmov se odloča za dvojiške nize vnaprej določene dolžine, kjer ima vsako mesto v kromosomu dve možni vrednosti: 0 in 1. Redkeje se za zapis kromosomov uporablja bolj naraven način, ki je odvisen od problema in namesto bitov uporablja prilagojene podatkovne strukture. Eden takšnih zapisov je naravni zapis z vektorji, kjer kromosom predstavimo z vektorjem. Tudi mi bomo uporabljali naravni zapis s podatkovnimi strukturami. Podrobneje si ga bomo ogledali v naslednjem razdelku. V tabeli 6 so zbrane prednosti in slabosti zapisa kromosomov v obliki dvojiškega niza in v naravni obliki.

	prednosti	**slabosti**
dvojiški niz	enostavna implementacija genetskih operatorjev	težje kodiranje in prepoznavanje rešitev, posebno dopustnih rešitev
naravna oblika	naravno kodiranje in prepoznavanje rešitev	zapletena implementacija genetskih operatorjev

Tabela 6: Prednosti in slabosti različnih zapisov kromosomov.

Selekcija. Ko smo se odločili za zapis, je naslednji korak izbira načina selekcije. Najprej moramo osebke oceniti. Za to uporabljamo **kriterijsko funkcijo**, ki vsakemu osebku priredi število – vrednost osebka. Kriterijska funkcija je odvisna od problema in njegovega zapisa.

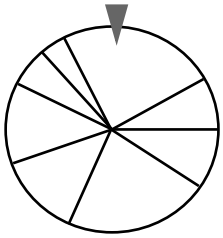
Namen selekcije je poudariti dobre osebke v upanju, da bodo njihovi potomci še boljši. Če je selekcija premočna, iz populacije izloči slabše osebke, ki so potrebni za raznolikost populacije. Če pa je selekcija prešibka, je evolucija prepočasna. Najti moramo pravi način selekcije, ki bo pripeljal do dobrih rešitev. V uporabi so naslednje metode selekcije:

1. **Selekcija z ruleto** (angl. roulette wheel selection)
   To je način selekcije, pri katerem je verjetnost, da bo osebek izbran, sorazmerna s kakovostjo osebka. To si lahko predstavljamo kot kolo, ki je razdeljeno na toliko delov, kolikor je osebkov v populaciji (slika 3). Velikost i-tega dela lahko izračunamo

kot koeficient

$$\frac{kakovost(i)}{\sum_i kakovost(i)}.$$

Če kolo zavrtimo, se bo ustavilo na določenem osebku z verjetnostjo, ki je sorazmerna z njegovo kakovostjo.



Slika 3: Selekcija z ruleto.

2. **Elitizem**
   Elitizem se uporablja kot dodatek k selekciji, pri katerem določeno število najboljših osebkov prestavimo v populacijo potomcev brez sprememb. Na tak način ne izgubimo najboljših rešitev. Elitizem se v mnogih genetskih algoritmih izkaže kot zelo učinkovit prijem za pospeševanje konvergence.

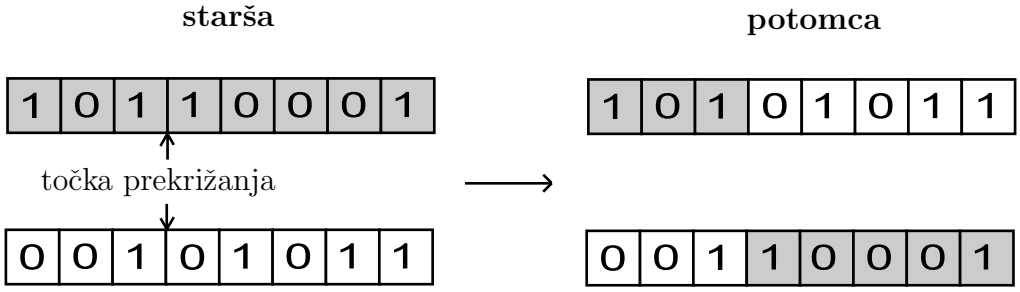
3. **Tekmovalna selekcija** (angl. tournament selection)
   Dva osebka iz populacije sta izbrana naključno. Nato naključno izberemo še število r med 0 in 1. Če je r manjši od določenega parametra (npr. 0.75), potem izberemo boljšega izmed osebkov, sicer pa slabšega. Osebka se nato vrneta v začetno populacijo in sta lahko še večkrat izbrana. S tekmovalno selekcijo preprečimo prehitro konvergenco algoritma.

4. **Počasna selekcija** (angl. steady-state selection)
   Večina genetskih algoritmov je generacijskih. To pomeni, da se osebki iz ene v drugo generacijo spremenijo v celoti ali vsaj večinoma (elitizem). Pri počasni selekciji pa se vsakič spremeni le nekaj slabših osebkov, ki jih zamenjajo najboljši potomci. Ta tip selekcije se uporablja v primerih, ko problem rešujejo vsi osebki skupaj in ne le posamezen najboljši osebek.

Genetski operatorji. Izbira genetskih operatorjev je močno pogojena z načinom zapisa problema. Najpogosteje uporabljani genetski operatorji so:

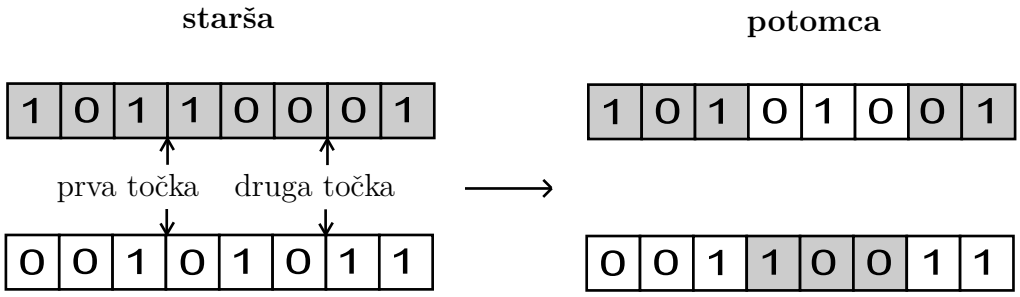
1. **Prekrižanje**

   Če imamo kromosome zapisane z dvojiškimi nizi, imamo na voljo več tipov prekrižanj. Najenostavnejše je prekrižanje z eno točko (slika 4). Naključno izberemo točko, od katere naprej si starša izmenjata ves zapis.



Slika 4: Prekrižanje z eno točko.

Slaba lastnost takega načina prekrižanja je, da ne doseže vseh mogočih rešitev z enako verjetnostjo. Zato je pogosto v uporabi tudi prekrižanje z dvema točkama (slika 5), pri katerem si starša izmenjata vse gene med obema točkama.



Slika 5: Prekrižanje z dvema točkama.

Pri dvojiškem zapisu kromosomov obstajajo tudi drugi načini prekrižanj, ki pa jih ne bomo opisovali. Če je kromosom zapisan v drugačni (naravni) obliki, je prekrižanje povsem odvisno od zapisa. Primer takšnega prekrižanja je prekrižanje, ki smo ga uporabili za naš problem urnika in je podrobneje opisano v naslednjem razdelku.

2. **Mutacija**

   Čeprav je prekrižanje glavno orodje, s katerim dosežemo izboljšanje rezultatov, je mutacija zelo uporabna zato, da se ne ustavimo v lokalnem optimumu problema.

Mutacija navadno spremeni naključni del kromosoma. Če so kromosomi zapisani v dvojiškem nizu, to pomeni, da se pri mutaciji naključni gen spremeni iz 0 v 1 (ali iz 1 v 0).

3. **Strategije parjenja**
   Poleg prekrižanj in mutacij posamezni raziskovalci uporabljajo tudi druge genetske operatorje. Zanimiv je pristop, ki pomaga ohranjati raznolikost osebkov v populaciji. To je dodajanje pogojev na parjenje. Na primer, če sta si dva osebka preveč podobna, parjenje prepovemo („incest").

Genetski algoritmi se med sabo razlikujejo tudi po tem, ali v postopku dovoljujejo nedopustne rešitve. Po delovanju genetskih operatorjev se namreč rešitve lahko „pokvarijo" – postanejo nedopustne. Spet imamo na razpolago več možnosti, kako se lotiti tega problema:

- Nekateri algoritmi v samih genetskih operatorjih poskrbijo za to, da rešitve ostajajo dopustne.
- Nekateri algoritmi dovolijo, da genetski operatorji naredijo rešitve nedopustne, te pa mora nato algoritem pred ocenjevanjem popraviti. To stori s t.i. **operatorjem *popravi***.
- Nekateri algoritmi dovolijo, da nedopustne rešitve obstajajo skozi celotno evolucijo. Take rešitve pri vrednotenju kakovosti označijo za zelo slabe.

Parametri. Da je genetski algoritem kar se da učinkovit, je treba najti pravo ravnotežje med selekcijo, prekrižanjem in mutacijo. To ravnotežje je odvisno tudi od kriterijske funkcije in oblike zapisa rešitve. Poleg tega se koristnost prekrižanj in mutacij s potekom genetskega algoritma spreminja. Zato je zelo pomembno, kakšne parametre izberemo. Med ključne parametre sodijo velikost populacije, verjetnost mutacije in podobno. Recept, ki bi povedal, kakšne parametre naj izberemo, ne obstaja. Navadno se parametre določa eksperimentalno in so od problema do problema drugačni. Grefenstette je leta 1986 prišel na zamisel, da bi za iskanje najprimernejših parametrov za genetski algoritem lahko uporabljali genetski algoritem!

Eden izmed parametrov genetskega algoritma je tudi **kriterij zaustavitve algoritma**. To je parameter, ki nam pove, kdaj naj končamo evolucijo. Najbolj pogosto uporabljeni kriteriji zaustavitve algoritma so naslednji:

- V evoluciji naredimo vnaprej določeno število generacij. Rešitev algoritma je najboljša rešitev zadnje generacije.
- Postavimo zahtevo aproksimacije, ki jo mora rešitev doseči, da se algoritem lahko konča. Če problem aproksimiramo samo z ene strani, je treba vsaj približno vedeti, kolikšna je optimalna rešitev problema.
- Razvijamo populacije in se ustavimo, ko se najboljša poznana rešitev v zadnjih nekaj korakih ni več spremenila.

To je bil kratek opis splošnih lastnosti genetskih algoritmov. Nadaljevali bomo z bolj podrobnim opisom našega genetskega algoritma za problem urnika.

3.2 Genetski algoritem za problem šolskega urnika

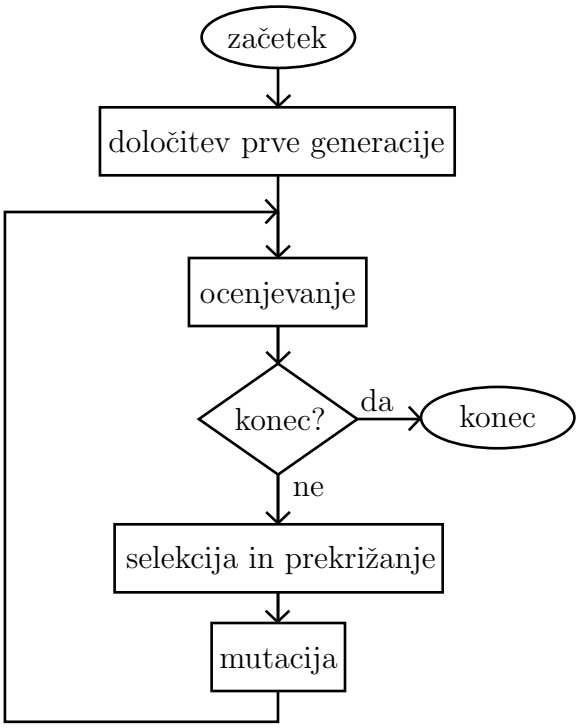
V prejšnjem razdelku smo spoznali glavne značilnosti genetskih algoritmov. Med drugim smo omenili tudi to, da ni vedno jasno, pri katerih problemih se bo genetski algoritem dobro odrezal. V zadnjem času je bilo veliko poskusov uporabe genetskih algoritmov na problemu šolskega urnika [4, 5, 7, 9, 11, 12, 15, 21, 22, 26, 27, 28]. Večina avtorjev se lahko pohvali z dobrimi rezultati in hitrimi algoritmi. Vsaka od omenjenih skupin avtorjev pa raziskuje in razvija svojo različico genetskega algoritma. To je razumljivo, saj smo že povedali, da je težko narediti rešitev za problem urnika, ki bi bila dobra za več različnih ustanov.

Tudi mi si bomo ogledali novo različico genetskega algoritma za problem urnika, ki po načinu zapisa podatkov še najbolj spominja na Carrascojevo rešitev iz [9]. Je pa rešitev sicer nova in preizkušena na podatkih z Oddelka za matematiko Fakultete za matematiko in fiziko Univerze v Ljubljani.

3.2.1 Potek algoritma

Na sliki 6 je narisan diagram poteka za naš genetski algoritem. V grobem ga lahko razdelimo na naslednje dele:

1. **Določitev prve generacije**
   Prva generacija je sestavljena iz dopustnih rešitev za dani problem urnika. Dopustne rešitve dobimo tako, da predavanjem dodelimo naključne predavalnice in ure ter se pri tem držimo strogih omejitev. Prva generacija tako nima kakovostnih rešitev, kar pa ni pomembno, saj bomo rešitve izboljševali z algoritmom.



Slika 6: Shema genetskega algoritma.

2. **Ocenjevanje**
   Za ocenjevanje posameznih osebkov trenutne populacije uporabljamo kriterijsko funkcijo, ki je sestavljena iz šibkih omejitev.

3. **Končanje izvajanja algoritma**
   Algoritem ustavi evolucijo, ko je zadoščeno zaustavitvenemu kriteriju. To je lahko takrat, ko so osebki dovolj kvalitetni, v našem primeru pa se to zgodi, ko smo razvili vnaprej določeno število generacij. Seveda lahko algoritem prekinemo tudi prej. V vsakem primeru nam algoritem vrne najboljšo dobljeno rešitev do trenutka, ko smo algoritem končali.

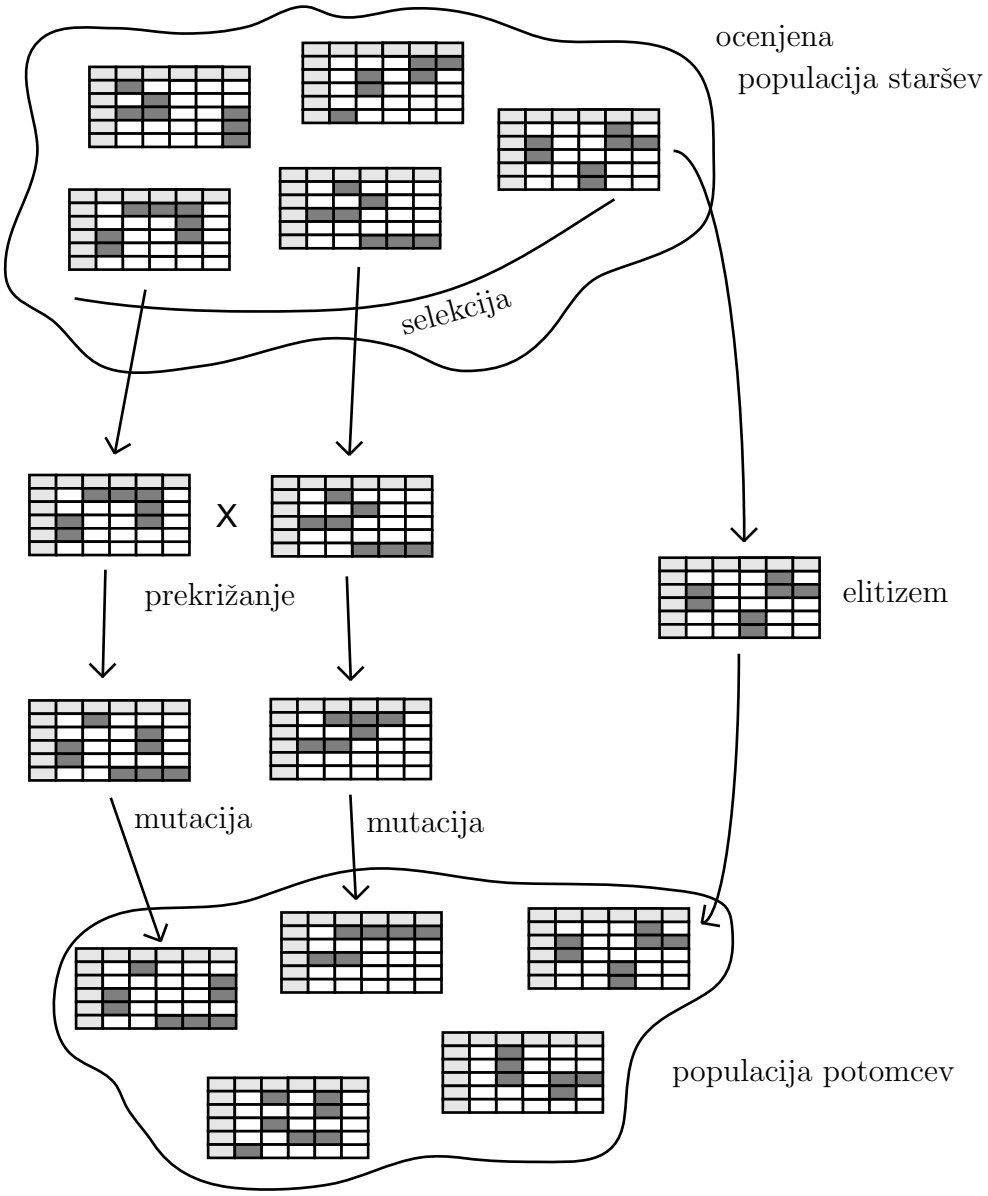
4. **Glavni del genetskega algoritma**
   Glavni del genetskega algoritma predstavlja postopek, ki iz populacije staršev naredi populacijo potomcev (slika 7). Na ocenjeni populaciji staršev najprej izvedemo selekcijo. Najboljši osebki nam predstavljajo elito, ki se brez prekrižanj in mutacij prepiše v populacijo potomcev. Nato do zapolnitve celotne populacije potomcev ponavljamo naslednje:

   (a) S selekcijo z ruleto izberemo dva starša.

(b) Starša prekrižamo, da dobimo dva potomca.

(c) Na potomcih uporabimo mutacijo in ju nato dodamo v populacijo potomcev.

Ko je populacija potomcev narejena, izvajanje algoritma nadaljujemo na točki 2.



Slika 7: Korak genetskega algoritma.

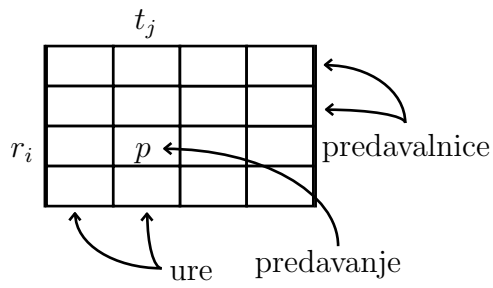
3.2.2 Zapis podatkov

Čeprav je pri genetskih algoritmih najpogosteje uporabljen zapis z dvojiškim nizom, je prav problem urnika eden tistih problemov, za katere ta zapis ni primeren. Večina

raziskovalcev problema urnika se namreč poslužuje naravnega zapisa s podatkovnimi strukturami. Uporabljeni naravni zapisi so od primera do primera praviloma različni, a imajo skupno prednost. Z naravnim zapisom je namreč nadzor nad posameznimi rešitvami lažji. V vsakem trenutku lahko enostavno izvemo, katere predavalnice so že zasedene, ali ima predavatelj ob nekem določenem terminu čas in podobno. Tako je mnogo lažje preverjati dopustnost rešitev, kar je pri zapisu z dvojiškim nizom še posebej težavna naloga.

Seveda pa ima zapis podatkov s podatkovnimi strukturami tudi svoje slabosti. Hitro se vidi, da je bolj zapleten in zahteva več računalniškega pomnilnika. Največja slabost naravnega zapisa pa se pokaže pri implementaciji genetskih operatorjev. Ta je neprimerno bolj zapletena od premikanja bitov pri zapisu z dvojiškimi nizi.

Pri zapisu osebka se bomo sklicevali na definiciji problema urnika $\mathcal{T}$ in njegove rešitve X, ki smo ju opisali v razdelku 2.2. Naj bo T množica vseh ur, R pa množica vseh predavalnic za problem urnika $\mathcal{T}$. Rešitev problema urnika (osebek v genetskem algoritmu) predstavimo z matriko, kjer vrstice pomenijo predavalnice, stolpci ure, v polja matrike pa vpisujemo predavanja. Če ima matrika v vrstici i in stolpcu j predavanje p, to pomeni, da se bo predavanje p izvajalo v predavalnici $r_i \in R$ ob uri $t_j \in T$ (slika 8). Zaradi takšne predstavitve bomo v nadaljevanju osebku včasih rekli kar matrika.



Slika 8: Osebek, zapisan v naravnem zapisu.

Dopustnost. V našem genetskem algoritmu morajo biti vsi osebki v populaciji dopustni. Osebkom, ki so še v izgradnji in ne vsebujejo vseh predavanj, pravimo *nepopolni osebki*. Nepopolni osebek je dopusten, če ni med vstavljenimi predavanji nobenih konfliktov. Dopustnost (nepopolnega) osebka vedno preverjamo že med njegovim nastajanjem, torej ko v matriko dodajamo predavanja. Pri tem ne smemo pozabiti, da pri vstavljanju večurnih predavanj zasedemo več zaporednih polj matrike. Predavanja morajo zadostiti vsem strogim omejitvam za vsa polja matrike, ki jih zapolnijo.

Na primeru si bomo ogledali, kako poteka sestavljanje enega osebka. Poseben poudarek dajemo na stroge omejitve, od katerih je odvisno, ali bo osebek dopusten.

Primer 3.1. Imamo problem urnika z naslednjimi podatki:

- Množico ur $T = \{t_1, \ldots, t_8\}$, ki so razdeljene na dva dneva. Perioda urnika je en „dvodnevni" teden. (Ta predpostavka se ne ujema s tisto v definiciji problema urnika, a jo bomo privzeli zaradi lažje predstavitve.)
- Množico predavalnic $R = \{r_1, r_2, r_3, r_4\}$ s kapacitetami $|r_1| = |r_2| = |r_3| = 50$ in $|r_4| = 100$. Predavalnica r_2 je ekskluzivna.
- Množico predavateljev $L = \{l_1, \ldots, l_5\}$.
- Množico skupin študentov $G = \{g_1, g_2, g_3\}$, ki so naslednjih velikosti: $|g_1| = |g_2| = 20$, $|g_3| = 60$.
- Množico predmetov $S = \{s_1, \ldots, s_6\}$, kjer je predmet s_4 edini, ki ima zahtevane predavalnice: $\eta(s_4) = \{r_2, r_4\}$.
- Množico prepovedanih ur za predavatelje $A = \{(l_3, t_6), (l_3, t_7)\}$.
- Množico prepovedanih ur za predavalnice $B = \{(r_2, t_1), (r_2, t_2), (r_4, t_5)\}$.
- Množico predavanj $P = \{p_1, \ldots, p_7\}$, ki so opisana v tabeli 7. (Predavanji p_6 in p_7 sta nastali iz iste skupine srečanj.)

predavanje	p_1	p_2	p_3	p_4	p_5	p_6	p_7
predavatelj	l_1	l_2	l_3	l_4	l_5	l_5	l_5
predmet	s_1	s_2	s_3	s_4	s_5	s_6	s_6
skupina	g_1	g_2	g_3	g_2	g_3	g_2	g_2
trajanje	3	2	1	2	3	1	1

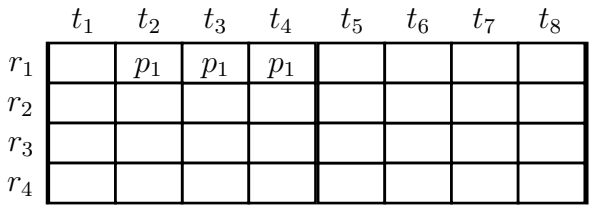
Tabela 7: Predavanja iz primera 3.1.

- Množico fiksnih razvrstitev $W = \{(p_1, r_1, t_2)\}$.

Nekatere pojme iz definicije problema urnika, kot so razredi in srečanja, smo tu namenoma izpustili, saj za prikazovanje tega primera niso pomembni.

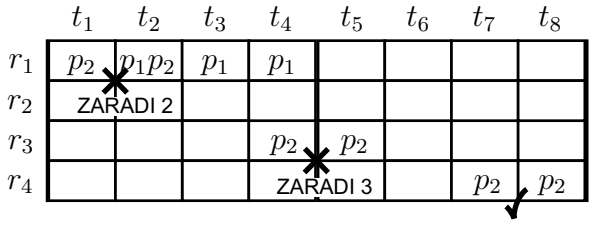
Naša naloga je, da v matriko, ki predstavlja osebek genetskega algoritma, vstavimo vsa predavanja iz množice P. Predavanja bomo vstavljali po vrsti, tako kot so napisana. Pri vstavljanju vsakega predavanja bomo s pomočjo strogih omejitev preverili, ali je trenutni nepopolni osebek še vedno dopusten:

1. *Fiksnih predavanj ne smemo premikati.*
   Ko gradimo osebek, najprej v matriko vstavimo vsa fiksna predavanja in tako dosežemo, da se ne bodo premikala. Tudi ta predavanja morajo upoštevati vse stroge omejitve.

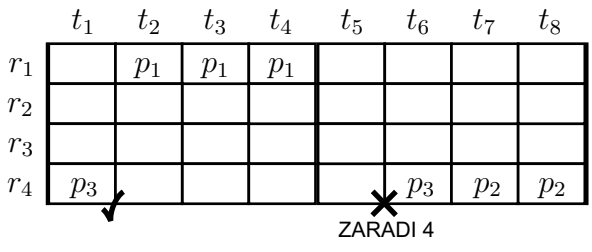


2. *Nobena predavalnica ne sme biti dodeljena več predavanjem hkrati.*
   Ta omejitev je izpolnjena zaradi izbire zapisa osebka, saj lahko v vsako polje matrike postavimo le eno predavanje.

3. *Predavanje se mora zgoditi v enem dnevu.*
   Na omejitev je treba paziti pri večurnih predavanjih, ki bi lahko segala še v naslednji dan.

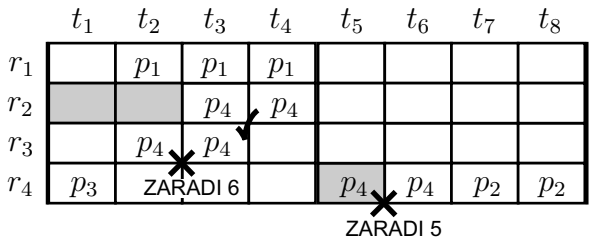


4. *Predavatelji imajo nekatere termine rezervirane za druge aktivnosti. Takrat jim ne smemo dodeliti predavanj.*
   Vsakič, ko želimo vstaviti neko predavanje v matriko, se moramo prepričati, ali ima predavatelj za vse ure trajanja predavanja čas.



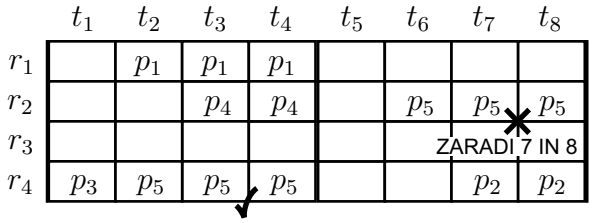
5. *Predavalnice so za nekatere termine nedostopne.*
   Termine, ob katerih so predavalnice zasedene, si lahko predstavljamo kot „črna" polja v matriki. V taka polja ne smemo postaviti nobenega predavanja.

6. *Upoštevati moramo zahtevane predavalnice.*
   Ko vstavljamo predavanje, katerega predmeti imajo zahtevane predavalnice, ga lahko vstavimo le v eno izmed zahtevanih predavalnic.



7. *Ekskluzivnim predavalnicam lahko dodelimo le predavanja, ki te predavalnice izrecno zahtevajo.*
   Predavanje, katerega predmeti nimajo zahtevanih predavalnic, lahko vstavimo le v neekskluzivne predavalnice.

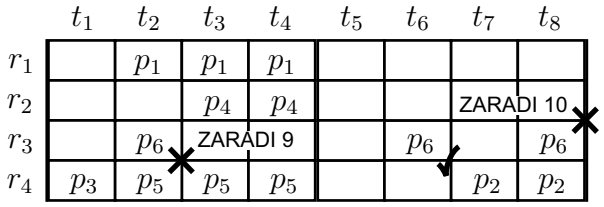
8. *Predavalnice morajo imeti dovolj prostora.*
   Za predavanje, ki ga želimo vstaviti, poznamo število študentov, ki ga bo poslušalo. Predavanje moramo uvrstiti v dovolj veliko predavalnico.



9. *Noben predavatelj ne sme imeti več predavanj hkrati.*
   Ta omejitev je enakovredna zahtevi, da ne sme noben predavatelj imeti dveh predavanj v istem stolpcu. Ko torej v matriko dodajamo novo predavanje, se moramo najprej prepričati, da predavatelj nima nobenega predavanja v stolpcih, ki jih bo zasedlo novo predavanje.

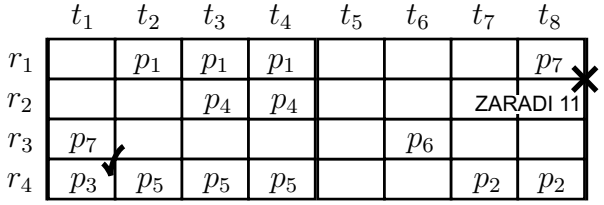
10. *Nobena skupina ne sme imeti več predavanj hkrati.*
    Podobno kot pri predavateljih moramo tudi tu za vsako skupino študentov posebej

preveriti, da nima predavanj v nobenem izmed stolpcev, v katere vstavljamo novo predavanje.



11. *Predavanja, ki nastanejo iz iste skupine srečanj, se lahko odvijajo le v različnih dnevih.*

    Ko vstavljamo predavanje v matriko, moramo preveriti, da ni v istem dnevu v matriki že kakšno predavanje iz iste skupine srečanj.



Ko v osebek vstavimo vsa predavanja in se pri tem držimo naštetih omejitev, dobimo dopustni osebek za naš algoritem. △

V primeru smo predvsem želeli pokazati, na katere stroge omejitve moramo biti pozorni pri vstavljanju novega predavanja v osebek. V našem algoritmu vstavljanje sicer poteka nekoliko drugače in je pri podprogramu *vstavi* natančneje opisano v nadaljevanju.

3.2.3 Selekcija

Kriterijska funkcija. Vrednost osebka v našem genetskem algoritmu je število, ki nam pove kakovost osebka. Večja vrednost vedno pomeni boljši osebek oz. boljši urnik. Vrednost osebka izračunamo s pomočjo kriterijske funkcije, ki je utežena funkcija naslednjih šibkih omejitev (podrobneje smo kriterijsko funkcijo že opisali na straneh 22–24):

1. *Predavanja naj se izvajajo ob urah, ki so najugodnejše za poučevanje.*
2. *Urnik naj vsebuje čim manj prostih ur za študente.*
3. *Študenti naj se čim manj selijo iz ene stavbe v drugo in iz ene predavalnice v drugo.*

Uteži, ki določajo, koliko bo posamezna šibka omejitev upoštevana, in kazni za selitve so parametri genetskega algoritma.

Elitizem. V našem genetskem algoritmu uporabljamo elitizem, ki najboljših nekaj osebkov iz starševske generacije prepiše neposredno v generacijo potomcev. Število osebkov v eliti je parameter genetskega algoritma.

Selekcija z ruleto. Kot način selekcije, ki iz populacije izbere dva osebka, smo izbrali selekcijo z ruleto. Selekcija z ruleto na vsakem koraku izbere dva starša z verjetnostjo, ki je sorazmerna z njuno vrednostjo. Starša ostaneta v starševski populaciji, kjer sta z enako verjetnostjo lahko ponovno izbrana.

3.2.4 Genetski operatorji

Tako prekrižanje kot mutacija morata v matriko, ki predstavlja osebek, vstavljati predavanja. Pri tem uporabljata podprogram *vstavi*, ki dano predavanje vstavi v osebek. Ker je ta podprogram bistveni del obeh operatorjev in ga uporabljamo tudi pri sestavljanju prve generacije, si ga bomo v nadaljevanju podrobneje ogledali.

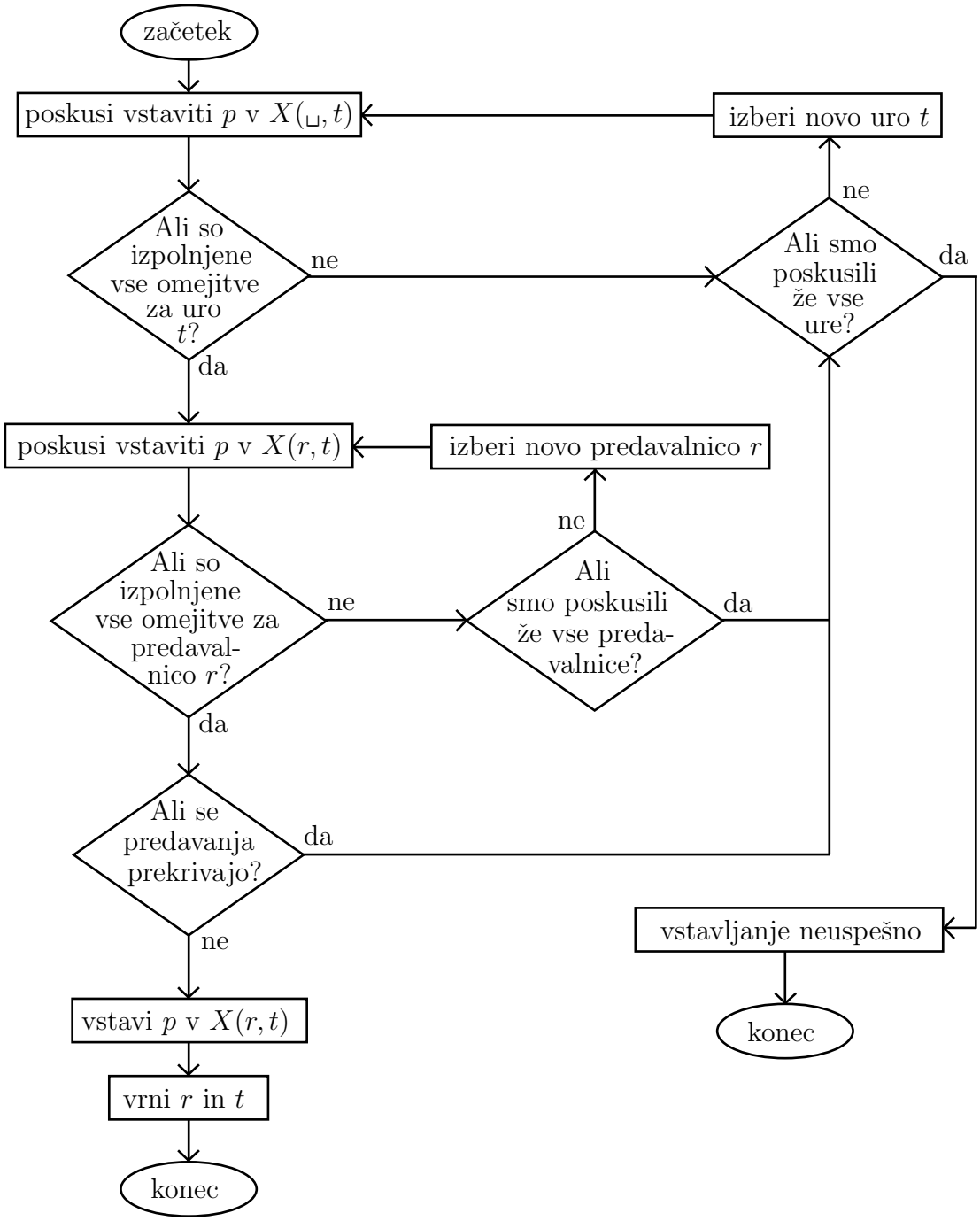
Podprogram *vstavi*. Podprogram dobi naslednje vhodne podatke: nepopolni osebek X, predavanje p, ki traja z ur, začetno uro t in predavalnico r. Naloga podprograma je, da poskuša vstaviti vse ure predavanja p v osebek X. Najprej poskuša predavanje vstaviti v predavalnico r od ure t do ure $t + z - 1$. Če to ni mogoče, preizkusi še druge ure in predavalnice, dokler ne izčrpa vseh možnosti.

Če vstavljanje ne uspe, potem predavanja p nikakor ne moremo vstaviti v osebek in sestavljanje osebka lahko prekinemo. Osebek tako ostane nepopolen, saj ne vsebuje vseh predavanj. Podprogram v takem primeru vrne vrednost, ki pomeni neuspešno vstavljanje.

Če vstavljanje uspe, podprogram vrne predavalnico in začetno uro, v katero smo vstavili predavanje. Osebek je narejen šele takrat, ko podprogram *vstavi* v osebek vstavi vsa predavanja.

Na sliki 9 je narisan diagram poteka, ki opisuje potek podprograma *vstavi*. Sledi krajše pojasnilo diagrama.

Podprogram bi rad vstavil predavanje p v matriko X na mesta $(r, t), \ldots, (r, t + z - 1)$ (na shemi je vedno označeno samo začetno mesto, torej (r, t)). Najprej preveri, ali lahko predavanje p sploh vstavi v ure $t, \ldots, t + z - 1$. Preveri torej vse stroge omejitve, ki so odvisne samo od ure. (V primeru 3.1 so to omejitve 3, 4 in 11.) Če kakšna omejitev ni izpolnjena, podprogram poskuša z novo uro. Če je vse ure že poskusil, vstavljanje ni uspelo in podprogram se konča.



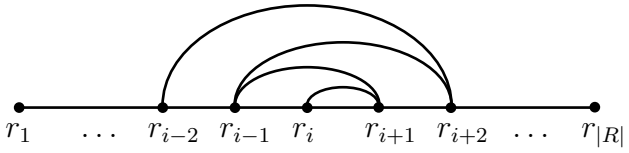
Slika 9: Shematični prikaz delovanja podprograma *vstavi*.

Če je bilo vsem omejitvam za ure zadoščeno, podprogram preveri, ali bo lahko vstavil predavanje na mesta $(r, t), \ldots, (r, t + z - 1)$ v matriki. Najprej preveri omejitve za predavalnico r v urah $t, \ldots, t + z - 1$. (V primeru 3.1 so to omejitve 2, 5, 6, 7 in 8.) Če so vse izpolnjene, preveri še, ali se predavanje prekriva s kakšnimi drugimi predavanji,

ki so že v matriki. (Omejitvi 9 in 10.) Če je tudi tu vse v redu, podprogram vstavi predavanje v matriko in konča.

Če katera od zahtev za predavalnico r ni bila izpolnjena, podprogram poskuša nadaljevati z novo predavalnico (če taka predavalnica še obstaja, sicer poskusi z drugo uro). Če bi se predavanje p prekrivalo s kakšnim drugim predavanjem, pa podprogram zamenja uro (spet samo v primeru, da obstaja še kakšna nepreizkušena ura).

Izbiranje novih predavalnic in ur ne poteka naključno. Predavalnice so urejene po velikosti – kapaciteti. Za vse $i = 1, \ldots, |R| - 1$ velja, da je $|r_i| \leq |r_{i+1}|$. Recimo, da v podprogramu *vstavi* želimo predavanje vstaviti v predavalnico r_i. Če to ne gre, moramo poskusiti z novo predavalnico. Katera bo, nam pove funkcija ***oscilacija***, ki predavalnice od r_i naprej izbira po vzorcu, ki ga prikazuje slika 10:



Slika 10: Izbira novih predavalnic z *oscilacijo*.

Torej je zaporedje predavalnic, ki jih bo podprogram *vstavi* poskusil, naslednje:

$$r_i, r_{i+1}, r_{i-1}, r_{i+2}, r_{i-2}, \ldots$$

Ko z oscilacijo pridemo nekje do konca (npr. do $r_{|R|}$), na drugi strani vzamemo po vrsti vse ure do drugega konca (npr. $r_j, r_{j-1}, \ldots, r_2, r_1$).

Izbiranje ur je nekoliko drugačno. Tu uporabljamo „uteženo" oscilacijo, ki izbira ure tako, da pridejo na vrsto najprej ure z večjo utežjo. Na ta način dosežemo, da se predavanja v naslednjih generacijah premikajo na bolj zaželene ure.

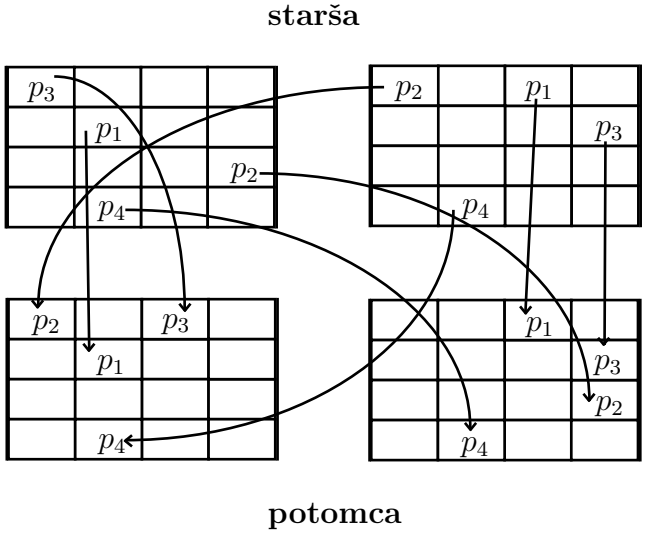
Prekrižanje. S prekrižanjem iz dveh osebkov iz populacije staršev dobimo dva potomca. S pomočjo slike 11 si bomo ogledali prekrižanje v našem naravnem zapisu.

Pri prekrižanju najprej oštevilčimo predavanja v obeh starših. (Predavanje p_i v materi je enako predavanju p_i v očetu za vsak i.) Oštevilčevanje je „skoraj naključno", saj imajo (zaradi pomanjkanja dovolj velikih predavalnic) prednost predavanja, ki imajo veliko število študentov. Skupine z enako študenti se pri vsakem prekrižanju izbirajo v naključnem vrstnem redu. Nato za vsako predavanje po vrsti ponavljamo naslednje korake:

1. Najprej naključno izberemo, kateri potomec bo podedoval položaj predavanja od matere in kateri od očeta. To naredimo s pomočjo parametra, ki predstavlja verjetnost pri prekrižanju. Če ima ta parameter vrednost ζ, mi pa naključno izberemo število med 0 in 1, ki je manjše od ζ, potem bo prvi potomec dedoval položaj od matere in drugi od očeta. Sicer bo ravno obratno.

2. Zdaj vemo, v katero polje matrike prvega potomca bi radi vstavili predavanje. Poskusimo ga vstaviti s pomočjo podprograma *vstavi*. Ker potomec deduje od obeh staršev, se lahko zgodi, da podprogram *vstavi* vstavi predavanje drugam, kot je bilo mišljeno (predavanje p_3 na sliki 11). Tudi na ta način pridobivamo nove kombinacije potomcev. Če vstavljanje uspe, nadaljujemo z naslednjim predavanjem, sicer pa prvega potomca ne gradimo več.

3. Isto naredimo tudi za drugega potomca.



Slika 11: Prekrižanje v izbranem naravnem zapisu.

Korake ponavljamo, dokler sta oba potomca še v izgradnji in še nismo vstavili vseh predavanj. Ko prekrižanje končamo, lahko dobimo dva, enega ali nobenega potomca. Potomce, ki smo jih dobili, obdelamo še z mutacijo in jih nato dodamo v naslednjo generacijo.

Mutacija. Mutacija je genetski operator, ki zagotavlja raznolikost populacije. Pri mutaciji si pomagamo s parametrom, ki določa verjetnost mutacije. To ni verjetnost,

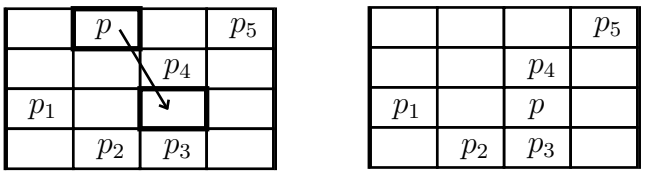
da bomo uporabili mutacijo na določenem osebku, ampak parameter, ki pove, s kolikšno verjetnostjo bomo uporabili mutacijo na vsakem predavanju v osebku. Potek mutacije lahko opišemo na naslednji način:

Po vrsti po vrsticah matrike za vsako predavanje v osebku naredimo naslednje:

1. Izberemo naključno število med 0 in 1 in ga primerjamo z verjetnostjo mutacije. Če je izbrano število manjše od verjetnosti mutacije, postopek nadaljujemo pri naslednji točki, sicer poskusimo z naslednjim predavanjem.

2. Naključno izberemo položaj v matriki. Ločimo primera, ko je položaj prazen in ko je zaseden s predavanjem:

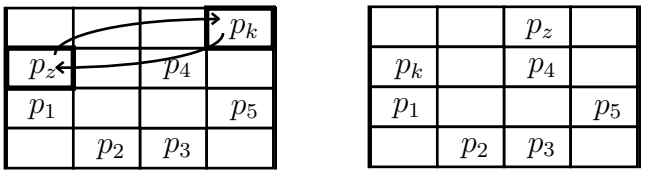
   (a) Naključno izbrani položaj v matriki je prazen (slika 12). Mutacija s pomočjo podprograma *vstavi* poskuša vstaviti predavanje na izbrani položaj. Če vstavljanje uspe (tudi če se premakne na kak drug položaj), nadaljujemo v točki 1 z naslednjim predavanjem. Sicer predavanje pustimo na prvotnem položaju in prav tako nadaljujemo z naslednjim predavanjem.



osebek | osebek po mutaciji predavanja p

Slika 12: Prvi primer pri mutaciji.

   (b) Na naključno izbranem položaju je že neko predavanje (slika 13). Zaradi lažjega razumevanja bomo govorili o začetnem (p_z) in končnem (p_k) predavanju. Začetno je tisto, ki smo ga izbrali na začetku, končno pa tisto, ki zaseda



osebek | osebek po mutaciji predavanja p_z

Slika 13: Drugi primer pri mutaciji.

naključno izbrano polje. Predavanji najprej vzamemo iz osebka zato, da sta položaja prazna. Spet uporabimo podprogram *vstavi*, ki poskuša začetno predavanje postaviti na položaj končnega predavanja. (To vstavljanje gotovo uspe, saj začetno predavanje vedno lahko postavimo nazaj na svoje mesto.) Nato poskušamo končno predavanje postaviti na položaj začetnega predavanja. Če vstavljanje ne uspe, moramo nazaj na izhodiščna položaja vrniti obe predavanji.

3.2.5 Parametri

V genetskem algoritmu je veliko različnih parametrov, ki vplivajo na njegovo učinkovitost:

- velikost populacije (število osebkov v populaciji),
- število osebkov v eliti,
- uteži šibkih omejitev,
- verjetnost pri prekrižanju in
- verjetnost mutacije.

K parametrom lahko štejemo tudi število vseh generacij, ki v našem primeru služi kot zaustavitveni kriterij algoritma.

Kako se bo algoritem odrezal pri reševanju problema urnika, je v veliki meri odvisno prav od teh parametrov. Več o vplivu parametrov na izvajanje algoritma in o najboljšem izboru parametrov bomo napisali v razdelkih 4.2 in 4.3.

4 Praktična izvedba genetskega algoritma s programom Kronos

V tem poglavju bomo prikazali, kako smo s programom Kronos implementirali genetski algoritem za problem šolskega urnika. Najprej bomo na kratko opisali zgradbo in delovanje programa. V nadaljevanju bomo razložili pomen parametrov genetskega algoritma in pokazali, kako različne vrednosti parametrov vplivajo na rešitve, ki jih zgradi algoritem. Na koncu bomo prikazali nekaj urnikov, dobljenih s programom Kronos.

4.1 Opis programa

Program Kronos je program, ki rešuje problem šolskega urnika z genetskim algoritmom, opisanim v prejšnjem poglavju. Program je napisan v jeziku C++ z orodjem Borland C++ Builder.

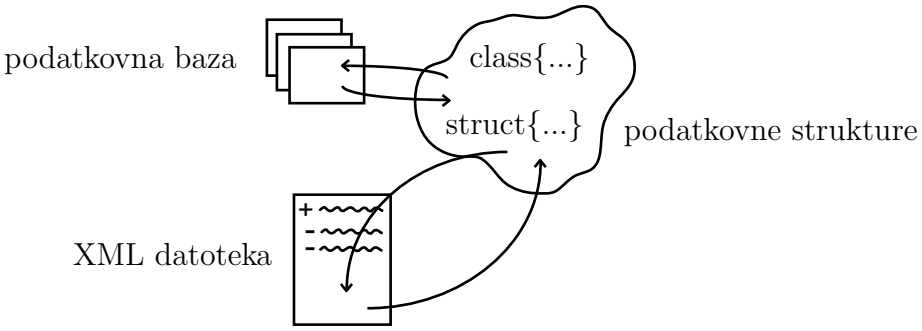
Program bomo opisali tako, da bomo najprej povedali, kakšni so njegovi vhodni in izhodni podatki. Nato si bomo ogledali podatkovne strukture, ki jih program uporablja za zapis problema in njegovo reševanje. Na koncu bomo še pokazali, kako se program uporablja.

4.1.1 Vhodni in izhodni podatki

Vhodni podatki programa Kronos so vsi podatki, ki definirajo problem urnika (glej definicijo šolskega urnika na strani 12). Izhodni podatek pa je rešen problem urnika oz. množica razvrstitev predavanj (glej definicijo rešitve problema urnika na strani 16).

V programu Kronos smo implementirali poenostavljen problem urnika, pri katerem je perioda urnika dolga en teden.

Vhodne podatke lahko program dobi na dva načina. Lahko jih prebere iz podatkovne baze ali iz datoteke tipa XML (eXtensible Markup Language). Tudi izhodne podatke lahko vpiše v podatkovno bazo ali v XML datoteko (slika 14).



Slika 14: Branje in pisanje podatkov v programu Kronos.

Navadno se v podatkovno bazo zapiše dokončen urnik, medtem ko se poskusni urniki shranjujejo le v XML datoteke. Pri branju podatkov pa je vseeno, ali jih beremo iz baze ali iz XML datoteke. Prednost branja podatkov iz XML datoteke je v tem, da lahko program Kronos poganjamo tudi na računalnikih, ki nimajo dostopa do podatkovne baze.

K diplomskemu delu je priložena različica programa Kronos, ki za zapis vhodnih in izhodnih podatkov uporablja le XML datoteke, zato bomo v nadaljevanju to obliko zapisa na kratko predstavili.

XML datoteke. XML je označevalni jezik (angl. markup language), s katerim lahko enostavno izdelamo nov označevalni jezik. Podatki v takšni datoteki so strukturirani in hierarhično urejeni. V programu Kronos smo to glavno funkcijo jezika XML zanemarili in smo XML datoteke uporabili le za zapis podatkov za problem urnika. V ta namen bi načeloma lahko uporabili katerikoli tekstovni zapis, prednost zapisa v obliki XML pa je naravna drevesna struktura, ki nam olajša delo. Zato na tem mestu ne bomo natančneje opisovali splošnih lastnosti jezika XML (več o XML datotekah lahko najdete v [14]), ampak bomo pozornost usmerili v sintakso, ki smo jo uporabili za zapis podatkov.

XML datoteke hranijo podatke v t.i. *elementih*. Vsak element je sestavljen na naslednji način: `<privesek>vsebina</privesek>`. Začetni in končni privesek oklepata vsebino. Elementi so lahko tudi brez vsebine. Tedaj namesto začetnega in končnega priveska uporabimo le en privesek: `<privesek/>`. Vsak element ima lahko tudi eno ali več *lastnosti*, ki jih zapišemo na naslednji način:

```
<privesek lastnost1="lep" lastnost2="primer">vsebina</privesek>.
```

Vsebina elementa je lahko nov element. Na ta način dobimo drevesno strukturo gnezdenih podatkov.

Primer zapisa podatkov v XML datoteki sledi na strani 57, kjer bomo pokazali, kako v jeziku XML zapišemo podatke o dveh predavanjih.

4.1.2 Uporabljene podatkovne strukture

Vse informacije, ki jih dobimo iz vhodnih podatkov, shranimo v podatkovne strukture programa. Nato ves čas izvajanja programa delamo le z njimi. Po končanem genetskem algoritmu lahko rešitev problema urnika iz podatkovnih struktur znova prepišemo v XML datoteko.

Podatkovne strukture so razdeljene na tri dele. Prvi del predstavlja podatke o problemu urnika. V drugem so podatki, ki jih potrebujemo pri izvajanju genetskega algoritma, torej podatki o populacijah, osebkih ipd. Tretji del podatkovnih struktur pa se uporablja pri grafičnem vmesniku programa. V nadaljevanju bomo podrobno opisali prvi in drugi del našega podatkovnega modela.

Ker je program napisan v objektno orientiranem jeziku C++, so podatkovne strukture, ki jih uporabljamo, *razredi* (angl. class) oz. *strukture* (angl. structure, skrajšano struct). Za vsako uporabljeno podatkovno strukturo bomo našteli njene pomembnejše podatkovne komponente in metode.

Podatki o problemu urnika. K podatkom o problemu urnika spadajo vsi podatki iz definicije problema urnika. Iz teh podatkov zgradimo naslednje razrede (slika 15):

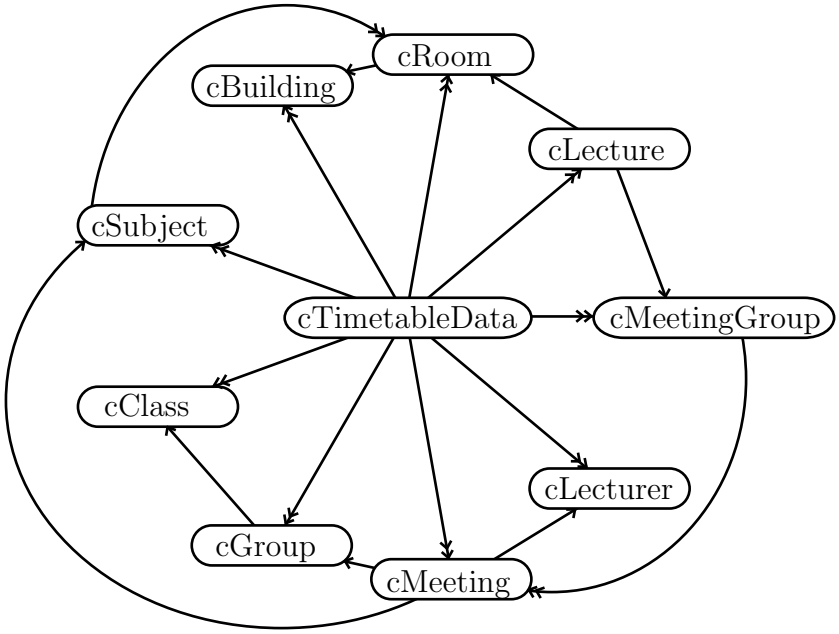
- **Razred:** `cObject`, **okrajšava:** `ob`
  Opis: To je abstrakten razred, iz katerega je izpeljana večina drugih razredov.
  Komponente:
  - `intIndexTTD` je indeks tega objekta v pripadajočem seznamu objektov `cObjectList`, ki ga vsebuje `cTimetableData`.
  - `strName` je niz znakov, ki opisuje objekt.
- **Razred:** `cObjectList`, **okrajšava:** `obl`
  Opis: To je abstrakten razred, ki združuje objekte tipa `cObject` v seznam (implementiran kot tabela).
  Komponenti:

– `intCount` je celo število, ki pove, koliko objektov je v seznamu.

– `obObjects` je seznam objektov tipa `cObject`.

Legenda:

A ⟶ B	A vsebuje en kazalec na B
A ⟶» B	A vsebuje vektor kazalcev na B



Slika 15: Povezava med razredi, ki vsebujejo podatke o urniku.

- **Razred:** `cTimetableData`, **okrajšava:** `ttd`
  Opis: To je glavni razred, ki vsebuje vse podatke za problem urnika.
  Komponente:

  – `intPeriodLength` je celo število, ki pomeni število tednov v periodi urnika.

  – `intDayHours` je celo število, ki pomeni število ur v enem dnevu.

  – `intNumHours` je celo število, ki pomeni število ur v periodi urnika.

  – `oblBuildings` je kazalec na objekt tipa `cObjectList`, v katerem so naštete vse stavbe (`cBuilding`).

  – `oblRooms` je kazalec na objekt tipa `cObjectList`, v katerem so naštete vse predavalnice (`cRoom`).

  – `oblLecturers` je kazalec na objekt tipa `cObjectList`, v katerem so našteti vsi predavatelji (`cLecturer`).

- `oblClasses` je kazalec na objekt tipa `cObjectList`, v katerem so našteti vsi razredi (`cClass`).
- `oblGroups` je kazalec na objekt tipa `cObjectList`, v katerem so naštete vse skupine študentov (`cGroup`).
- `oblSubjects` je kazalec na objekt tipa `cObjectList`, v katerem so našteti vsi predmeti (`cSubject`).
- `oblMeetings` je kazalec na objekt tipa `cObjectList`, v katerem so našteta vsa srečanja (`cMeeting`).
- `oblMeetingGroups` je kazalec na objekt tipa `cObjectList`, v katerem so naštete vse skupine srečanj (`cMeetingGroup`).
- `oblLectures` je kazalec na objekt tipa `cObjectList`, v katerem so našteta vsa predavanja (`cLecture`).

Metodi: Metoda `ReadXMLData` prebere podatke iz XML datoteke in jih shrani v komponente, metoda `WriteXMLData` pa podatke iz komponent razreda zapiše v XML dokument.
Opomba: V podatkovnem modelu niso nikjer eksplicitno zapisane ure. Vemo le, da jih je `intNumHours`. Ker gredo ure po vrsti od 0 do `intNumHours - 1`, na ta način ure prepoznavamo. Npr. v seznamu `blFreeHours` nekega predavatelja je `blFreeHours[i]` vrednost, ki pove, ali je predavatelj prost ob uri i.

- **Razred:** `cBuilding`, **okrajšava:** `bg`
  Opis: Razred predstavlja stavbo, v kateri so predavalnice. Izpeljan je iz razreda `cObject`.

- **Razred:** `cRoom`, **okrajšava:** `rm`
  Opis: Razred predstavlja eno predavalnico. Izpeljan je iz razreda `cObject`.
  Komponente:
  - `intCapacity` je celo število, ki predstavlja zmogljivost predavalnice.
  - `blExclusive` je logična spremenljivka, ki pove, ali je predavalnica ekskluzivna.
  - `bgOwner` je kazalec na stavbo (`cBuilding`), v kateri se nahaja predavalnica.
  - `blFreeHours` je tabela logičnih vrednosti, ki za vsako uro pove, ali je predavalnica takrat dosegljiva.

- **Razred:** `cLecturer`, **okrajšava:** `lr`
  Opis: Razred predstavlja predavatelja. Izpeljan je iz razreda `cObject`.
  Komponenta: `blFreeHours` je tabela logičnih vrednosti, ki za vsako uro pove, ali ima predavatelj takrat čas.

- **Razred:** `cClass`, **okrajšava:** `cl`
  Opis: Razred predstavlja razred študentov. Izpeljan je iz razreda `cObject`.
  Komponenta: `intHourWeights` je tabela celih števil, ki pomenijo uteži za ure razreda.

- **Razred:** `cGroup`, **okrajšava:** `gr`
  Opis: Razred predstavlja skupino študentov. Izpeljan je iz razreda `cObject`.
  Komponenti:
  - `intNumStudents` je celo število, ki pove število študentov v skupini.
  - `clClass` je kazalec na objekt tipa `cClass`, v katerega spada skupina.

- **Razred:** `cSubject`, **okrajšava:** `sb`
  Opis: Razred predstavlja predmet. Izpeljan je iz razreda `cObject`.
  Komponenti:
  - `flWeekHours` je racionalno število, ki pove, koliko ur na teden mora imeti predmet.
  - `oblRequiredRooms` je kazalec na objekt tipa `cObjectList`, v katerem so naštete vse zahtevane predavalnice (`cRoom`) predmeta.

- **Razred:** `cMeeting`, **okrajšava:** `mt`
  Opis: Razred predstavlja srečanje. Izpeljan je iz razreda `cObject`.
  Komponente:
  - `intPeriodLength` je celo število, ki pomeni število tednov v periodi srečanja.
  - `intNumHours` je celo število, ki pomeni trajanje srečanja.
  - `intMultiplicity` je celo število, ki pomeni večkratnost srečanja.
  - `sbSubject` je kazalec na predmet (`cSubject`), ki se izvaja v srečanju.
  - `grGroup` je kazalec na skupino (`cGroup`), ki sodeluje v srečanju.
  - `lrLecturer` je kazalec na predavatelja (`cLecturer`), ki predava v srečanju.

- **Razred:** `cMeetingGroup`, **okrajšava:** `mg`
  Opis: Razred predstavlja skupino srečanj. Izpeljan je iz razreda `cObject`.
  Komponenti:
  - `intNumStudents` je celo število, ki pove, koliko je vseh študentov, ki bodo sodelovali v skupini srečanj.
  - `oblMeetings` je kazalec na objekt tipa `cObjectList`, v katerem so našteta vsa srečanja (`cMeeting`), ki sestavljajo skupino srečanj.

- **Razred:** `cLecture`, **okrajšava:** `lt`
  Opis: Razred je najpomembnejši razred v podatkovnem modelu, saj predstavlja predavanje in njegovo razvrstitev. Izpeljan je iz razreda `cObject`.
  Komponente:
  - `mgMeetingGroup` je kazalec na skupino srečanj (`cMeetingGroup`), katere del je to predavanje.
  - `intMgrPart` je celo število, ki pove, kateri del skupine srečanj je to predavanje.
  - `intStartHour` je celo število, ki pomeni uro začetka predavanja.
  - `intNumHours` je celo število, ki pomeni število ur trajanja predavanja.
  - `rmRoom` je kazalec na predavalnico (`cRoom`), v kateri se bo predavanje odvijalo.
  - `blFixed` je logična spremenljivka, ki pove, ali je predavanje bilo vnaprej fiksirano.

  Metode: Za vsako predavanje nam funkcija `GetLecturer` vrne njegovega predavatelja. Podobno nam funkcija `GetSubjects` vrne predmete, ki se predavajo. Funkcija `GetGroups` vrne skupine študentov, funkcija `GetClasses` pa razrede, ki sodelujejo na predavanju.
  Opomba: Na začetku spremenljivki `intStartHour` in `rmRoom` nimata določenih vrednosti (razen če je predavanje že fiksirano). Naloga programa oz. genetskega algoritma je določiti vrednosti teh dveh komponent, torej predavanju dodeliti začetno uro in predavalnico.

Vse naštete podatke program dobi iz vhodnih podatkov, ki so zapisani v XML datoteki. Na primeru si oglejmo, kako izgledajo vhodni podatki za predavanje.

Primer 4.1. Poglejmo si primer zapisa dveh predavanj v XML datoteki. Prvo predavanje še nima določene začetne ure in predavalnice, drugo predavanje pa je fiksno in ima

tako začetno uro kot predavalnico že določeno.

```
- <oblLectures>
  - <cLecture ID="616" intIndexTTD="95" intMgrPart="1"
    intStartHour="-1" intNumHours="2" blFixed="0">
      <obRoom ref="-1" />
      <mgMeetingGroup ref="266" />
  </cLecture>
  - <cLecture ID="617" intIndexTTD="96" intMgrPart="1"
    intStartHour="51" intNumHours="1" blFixed="1">
      <obRoom ref="23" />
      <mgMeetingGroup ref="268" />
  </cLecture>
</oblLectures>
```

Hitro lahko vidimo, da je zapis podatkov za predavanje v XML datoteki zelo podoben obliki podatkov, ki smo jo definirali z razredom `cLecture`. Tako lahko podatke hitro preberemo in jih shranimo v naš podatkovni model. △

Podatki o genetskem algoritmu. Da lahko na problemu urnika poženemo genetski algoritem, potrebujemo naslednje podatkovne strukture (slika 16), ki opisujejo podatke genetskega algoritma:

- **Razred:** `cConstraint`, **okrajšava:** `cn`
  Opis: To je abstraktni razred, iz katerega so izpeljani razredi, ki predstavljajo omejitve:

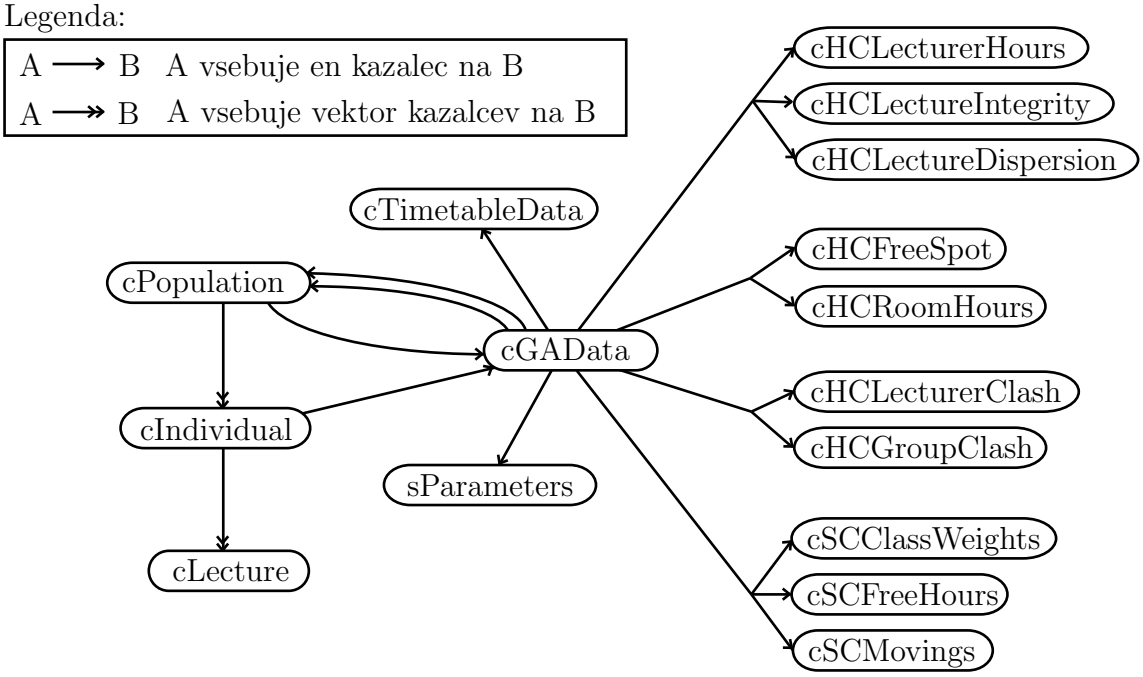
  Stroge omejitve:

  - `cHCLecturerHours` je razred, ki predstavlja omejitev, da moramo upoštevati predavateljeve proste ure.
  - `cHCLectureIntegrity` je razred, ki predstavlja omejitev, da mora predavanje potekati v enem dnevu.
  - `cHCLectureDispersion` je razred, ki predstavlja omejitev, da se dve predavanji iz iste skupine srečanj lahko predavata le v različnih dnevih.
  - `cHCFreeSpot` je razred, ki predstavlja omejitev, da je v eni predavalnici lahko samo eno predavanje.

– `cHCRoomHours` je razred, ki predstavlja omejitev, da moramo upoštevati proste ure predavalnic.

– `cHCLecturerClash` je razred, ki predstavlja omejitev, da ima predavatelj lahko le eno predavanje naenkrat.

– `cHCGroupClash` je razred, ki predstavlja omejitev, da imajo skupine lahko le eno predavanje naenkrat.



Slika 16: Povezava med razredi, ki vsebujejo podatke o genetskem algoritmu.

Šibke omejitve:

– `cSCClassWeights` je razred, ki predstavlja omejitev, da naj upoštevamo uteži za ure razredov.

– `cSCFreeHours` je razred, ki predstavlja omejitev, da naj imajo skupine študentov čim manj prostih ur.

– `cSCMovings` je razred, ki predstavlja omejitev, da naj se študenti čim manj selijo iz predavalnice v predavalnico in iz stavbe v stavbo.

Metodi: Razred ima dve navidezni funkciji, ki ju izpeljani razredi prekrijejo. To sta funkcija `InConflict`, ki pove, ali je osebek zaradi omejitve v konfliktu (funkcijo

prekrijejo razredi, ki predstavljajo stroge omejitve), in funkcija `Evaluate`, ki vrne vrednost osebka glede na šibko omejitev.

- **Razred:** `cConstraintList`, **okrajšava:** `cnl`
  Opis: To je abstraktni razred, ki združuje razrede `cConstraint` v seznam.
  Komponenti:
  - `intCount` je celo število, ki pove, koliko objektov je v seznamu.
  - `cnConstraints` je kazalec na seznam objektov tipa `cConstraint`.

- **Razred:** `cGAData`, **okrajšava:** `gad`
  Opis: To je glavni razred, ki vsebuje vse podatke za genetski algoritem.
  Komponente:
  - `intGeneration` je celo število, ki pove, katera po vrsti je trenutna generacija.
  - `popParent` je kazalec na starševsko populacijo (`cPopulation`).
  - `popOffspring` je kazalec na populacijo potomcev (`cPopulation`).
  - `ttdData` je kazalec na podatke o problemu urnika (`cTimetableData`).
  - `prm` je kazalec na parametre (`sParameters`).
  - `cnlHardHour` je kazalec na objekt tipa `cConstraintList`, v katerem so naštete vse stroge omejitve (`cConstraint`), ki so odvisne od ure.
  - `cnlHardHourRoom` je kazalec na objekt tipa `cConstraintList`, v katerem so naštete vse stroge omejitve (`cConstraint`), ki so odvisne od ure in predavalnice.
  - `cnlHardClash` je kazalec na objekt tipa `cConstraintList`, v katerem so naštete vse stroge omejitve (`cConstraint`), ki pregledujejo prekrivanje predavanj.
  - `cnlSoft` je kazalec na objekt tipa `cConstraintList`, v katerem so naštete vse šibke omejitve (`cConstraint`).

  Metode: Najpomembnejša metoda v tem razredu je gotovo podprogram `GeneticAlgorithm`, v katerem poteka evolucija. Med drugim razred vsebuje še metodo za urejanje števil (`Sort`) in dve metodi, ki vrneta permutacijo števil (uporabljamo ju pri prekrižanju). V metodi `GetConstraints` nastavimo vse stroge in šibke omejitve problema urnika.

- **Razred:** `cPopulation`, **okrajšava:** `pop`
  Opis: Razred predstavlja eno populacijo osebkov.
  Komponente:
  - `gadData` je kazalec na podatke o genetskem algoritmu (`cGAData`).
  - `oblIndividuals` je kazalec na objekt tipa `cObjectList`, v katerem so našteti vsi osebki v populaciji (`cIndividual`).
  - `oblElite` je kazalec na objekt tipa `cObjectList`, v katerem so našteti vsi elitni osebki v populaciji (`cIndividual`).
  - `inBest` je kazalec na najboljši osebek populacije (`cIndividual`).

  Metode: V metodi `Evaluate` ocenimo vse osebke v populaciji. V metodi `RouletteSelection` je implementirana selekcija z ruleto, v `Crossover` prekrižanje in v metodi `Mutation` mutacija.

- **Razred:** `cIndividual`, **okrajšava:** `in`
  Opis: Razred predstavlja en osebek.
  Komponente:
  - `gadData` je kazalec na podatke o genetskem algoritmu (`cGAData`).
  - `ltLecture[MAX_NUM_RM][MAX_NUM_HOURS]` je dvorazsežna tabela kazalcev na predavanja (`cLecture`). To je tabela, ki smo jo opisali pri zapisu podatkov na strani 39. `ltLecture[r][t]` je kazalec na predavanje, ki se začne ob uri t in poteka v predavalnici r.
  - `intRooms` je tabela indeksov predavalnic po indeksu predavanj. `intRooms[i]` je indeks predavalnice, v katero je postavljeno i-to predavanje.
  - `intHours` je tabela začetnih ur po indeksu predavanj. `intHours[i]` je začetna ura, v katero je postavljeno i-to predavanje.
  - `flValue` je vrednost osebka.

  Metode: V metodi `Evaluate` ocenimo osebek. Metoda `Insert` predstavlja podprogram *vstavi*, ki smo ga opisali na strani 45. V metodah `GetOscillation` in `GetWeightedOscillation` sta opisani oscilaciji (navadna in utežena).

- **Struktura:** `sInsertResult`, **okrajšava:** `ir`
  Opis: Struktura predstavlja rezultat podprograma *vstavi*.
  Komponente:

  - `intResult` je celo število, ki pove, ali je vstavljanje uspelo.
  - `intRoom` je celo število, ki pove indeks predavalnice, v katero je podprogram vstavil predavanje.
  - `intHour` je celo število, ki pove začetno uro, v katero je podprogram vstavil predavanje.

- **Struktura:** `sParameters`, **okrajšava:** `prm`
  Opis: V strukturi so zbrani vsi parametri genetskega algoritma.
  Komponente:

  - `intNUM_GEN` je celoštevilski parameter, ki pove število generacij v evoluciji.
  - `intNUM_IN` je celoštevilski parameter, ki pove število osebkov v eni generaciji.
  - `intNUM_ELITE` je celoštevilski parameter, ki pove število elitnih osebkov v eni generaciji.
  - `flWEIGHT_CLASSWEIGHTS` je utež za ure razredov.
  - `flWEIGHT_FREEHOURS` je utež za proste ure.
  - `flWEIGHT_MOVINGS` je utež za selitve študentov.
  - `flPENALTY_MOVINGS_BG` je kazen za selitve med stavbami.
  - `flPENALTY_MOVINGS_RM` je kazen za selitve med predavalnicami.
  - `intMUTATION_PROBABILITY` je celoštevilski parameter, ki pove verjetnost mutacije v promilih.
  - `intCROSSOVER_PROBABILITY` je celoštevilski parameter, ki pove verjetnost pri prekrižanju v odstotkih.

  Opomba: Utež `flWEIGHT_FREEHOURS` se pri računanju uteži za proste ure dodatno pomnoži z globalno definirano utežjo `EQUALIZE_WEIGHT`. Na ta način uravnovesimo šibki omejitvi za ure razredov in proste ure.

4.1.3 Uporaba programa

K diplomskemu delu je priložena zgoščenka z naslednjo vsebino:

- Z datoteko `Kronos.exe` poženemo program Kronos.

- Datoteka `Podatki.xml` vsebuje podatke za problem urnika, datoteka `Resitev.xml` pa eno boljših rešitev tega problema. Drevesna struktura podatkov iz teh datotek je najbolje vidna, če datoteki pregledujemo s kakšnim internetnim brskalnikom. Sicer pa lahko vsebino datotek vidimo tudi v kateremkoli tekstovnem urejevalniku.
- V datoteki `Navodila.txt` lahko preberemo navodila za uporabo programa.
- Poleg omenjenih datotek so na zgoščenki tudi datoteke s končnicama `dll` in `bpl`. To so dinamične knjižnice, ki jih program potrebuje za delovanje.
- V mapi `Statistika` se nahajajo datoteke, ki vsebujejo rezultate preizkušanja vpliva parametrov genetskega algoritma: `Stat_Osebki.csv` – število osebkov, `Stat_Elita.csv` – odstotek osebkov v eliti, `Stat_Mutacije.csv` – verjetnost mutacije in `Stat_Prekrizanje.csv` – verjetnost pri prekrižanju.

Zgoščenka vsebuje vse, kar potrebujemo za preprosto uporabo programa Kronos. Z njim si lahko ogledamo že narejen urnik (npr. tistega, ki je shranjen v datoteki `Resitev.xml`) ali pa sami naredimo novo rešitev za dani problem urnika.

Navodilo za uporabo programa:

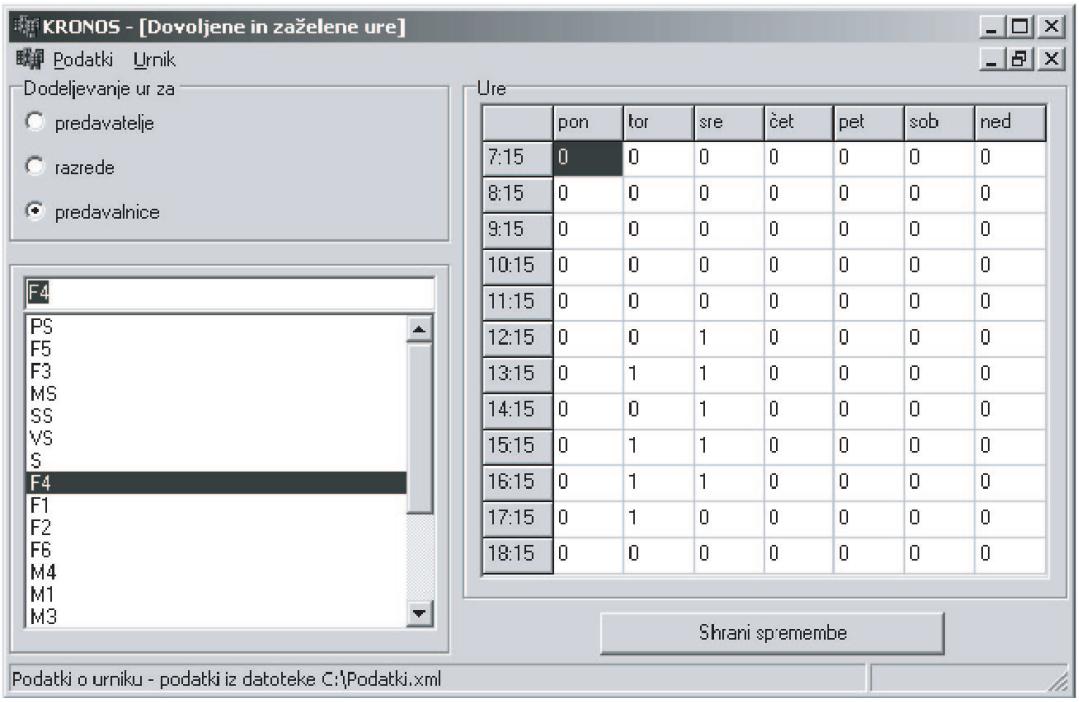
1. Program poženemo z datoteko **Kronos.exe**.
2. Najprej moramo naložiti podatke. To naredimo tako, da izberemo možnost **Podatki → Naloži podatke** v meniju programa. Odpre se okno, v katerem izberemo datoteko z urnikom. Nalaganje podatkov lahko traja nekaj sekund; program nas obvesti, ko je končano. V statusni vrstici na dnu okna programa je zapisano ime datoteke, iz katere smo naložili podatke.
3. Ko imamo podatke naložene, v meniju izberemo možnost **Urnik → Dodeljevanje ur**, s katero se nam odpre novo okno. V tem oknu lahko izberemo naslednje možnosti:

   (a) **Dodeljevanje ur predavateljem.**
   Če izberemo to možnost, se nam v seznamu prikažejo vsi predavatelji, ki jih razporejamo v urnik. S klikom na enega izmed njih se v tabeli prikažejo njegove dovoljene/prepovedane ure. Ure za predavatelja lahko spreminjamo z vnašanjem v tabelo. Vrednost 1 v celici tabele pomeni, da je ta ura za predavatelja dovoljena. Z vrednostjo 0 označimo prepovedane ure predavatelja.

Vse spremembe, ki jih naredimo v tabeli, se shranijo šele, ko pritisnemo na gumb **Shrani spremembe**.

(b) **Dodeljevanje ur razredom.**
Pri tej možnosti se nam v seznamu prikažejo vsi razredi, ki jih razporejamo v urnik. S klikom na enega izmed njih se v tabeli prikažejo uteži za ure razreda. Kot pri predavateljih tudi tu velja, da lahko uteži spreminjamo z vnašanjem novih vrednosti v tabelo. Uteži za razrede morajo biti nenegativna cela števila. Večja utež pomeni bolj zaželeno uro. Vse spremembe, ki jih naredimo v tabeli, se shranijo šele, ko pritisnemo na gumb **Shrani spremembe**.

(c) **Dodeljevanje ur predavalnicam.**
Če izberemo to možnost, se nam v seznamu prikažejo vse predavalnice, ki so na voljo (slika 17). S klikom na eno izmed njih se v tabeli prikažejo dovoljene/prepovedane ure za predavalnico. Postopek za spreminjanje ur je enak kot pri predavateljih. Tudi tu sta dovoljeni vrednosti za ure 0 in 1. Vse spremembe, ki jih naredimo v tabeli, se shranijo šele, ko pritisnemo na gumb **Shrani spremembe**.



Slika 17: Program Kronos: dodeljevanje ur predavalnicam.

S pritiskom na gumb **Shrani spremembe** vedno shranimo spremembe le v po-

datkovne strukture programa. Če želimo kasneje ponovno pognati generiranje urnika s pravkar izbranimi podatki, potem je dobro, da celotne podatke shranimo v XML datoteko. To naredimo tako, da v meniju izberemo **Podatki → Shrani podatke**. V oknu, ki se prikaže, izberemo ime datoteke, v katero želimo shraniti podatke. Program nas obvesti, ko je shranjevanje podatkov končano.

4. Ko smo zadovoljni z urami, lahko pričnemo z avtomatičnim generiranjem urnika. V meniju izberemo ukaz **Urnik → Generiranje urnikov**, ki nam odpre novo okno (slika 18). V oknu imamo vnosna polja za vse parametre genetskega algoritma. Vnosna polja že vsebujejo privzete vrednosti, ki pa jih lahko spremenimo.

   V polja **Število generacij**, **Število osebkov v generaciji** in **Število osebkov v eliti** vpišemo pozitivna cela števila. Število osebkov v eliti je lahko tudi 0, a mora biti manjše od števila vseh osebkov.

   V polje **Utež** napišemo utež, s katero se bodo pomnožile vse druge uteži. To je edina utež, ki je lahko realno število. Navadno je to zelo majhno število (npr. 0.0001), s katerim dosežemo, da vrednosti osebkov ostajajo „majhne" (npr. namesto da govorimo o milijonskih vrednostih, dobimo vrednosti v tisočih). Razložimo to na primeru:

   Recimo, da bi radi pognali program z naslednjimi vrednostmi: **Utež za razrede** $= 0.01$, **Utež za proste ure** $= 0.2$ in **Utež za selitve** $= 0.0001$. Vrednost glavne uteži je enaka redu najmanjše uteži, ostale uteži pa primerno povečamo. Tako dobimo: **Utež** $= 0.0001$, **Utež za razrede** $= 100$, **Utež za proste ure** $= 2000$ in **Utež za selitve** $= 1$.

   Utež za selitve ima še dva parametra: **Kazen za selitev med stavbami** in **Kazen za selitev med predavalnicami**. To sta kazni β_2 in β_1 iz primera 2.2. Zanju izberemo pozitivni celi števili.

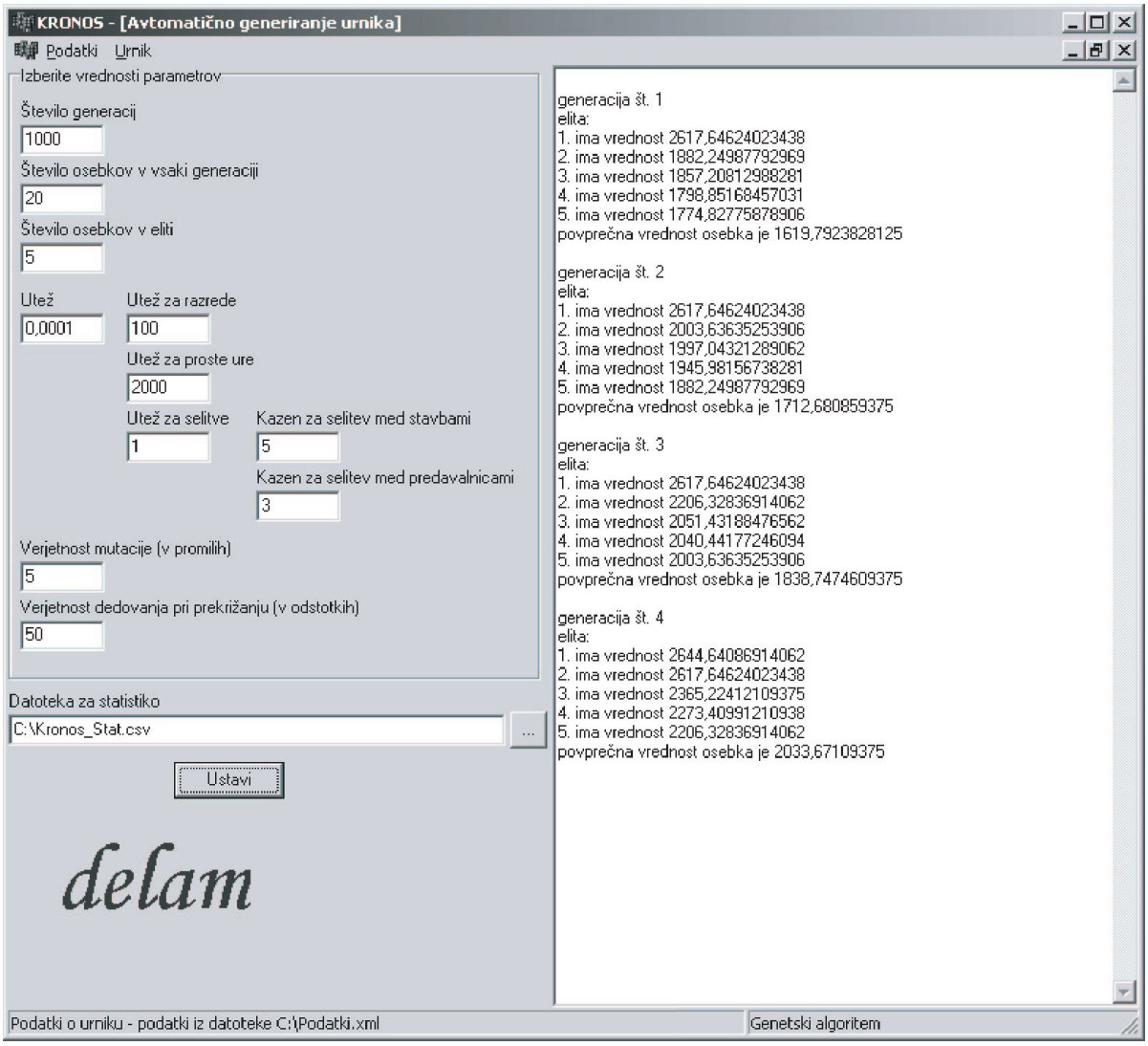
   Izbrati moramo tudi **Verjetnost mutacije v promilih** in **Verjetnost pri prekrižanju v odstotkih**.

   Ko smo izbrali vse parametre, v polje **Datoteka za statistiko** vpišemo pot in ime tekstovne datoteke tipa CSV (Comma Separated Values), v katero se bo med izvajanjem programa shranjevala statistika evolucije.

   S klikom na gumb **Začni** pričnemo z generiranjem urnika. Med izvajanjem algoritma se nam na desni strani okna izpisujejo osnovni podatki o posameznih generacijah. Tako lahko sledimo poteku evolucije in tudi predvidimo, koliko časa bo

algoritem potreboval, da se konča. V vsakem trenutku lahko prekinemo izvajanje algoritma s klikom na gumb **Ustavi**. Program bo takrat dokončal generacijo, ki je v delu, in zatem ustavil genetski algoritem.

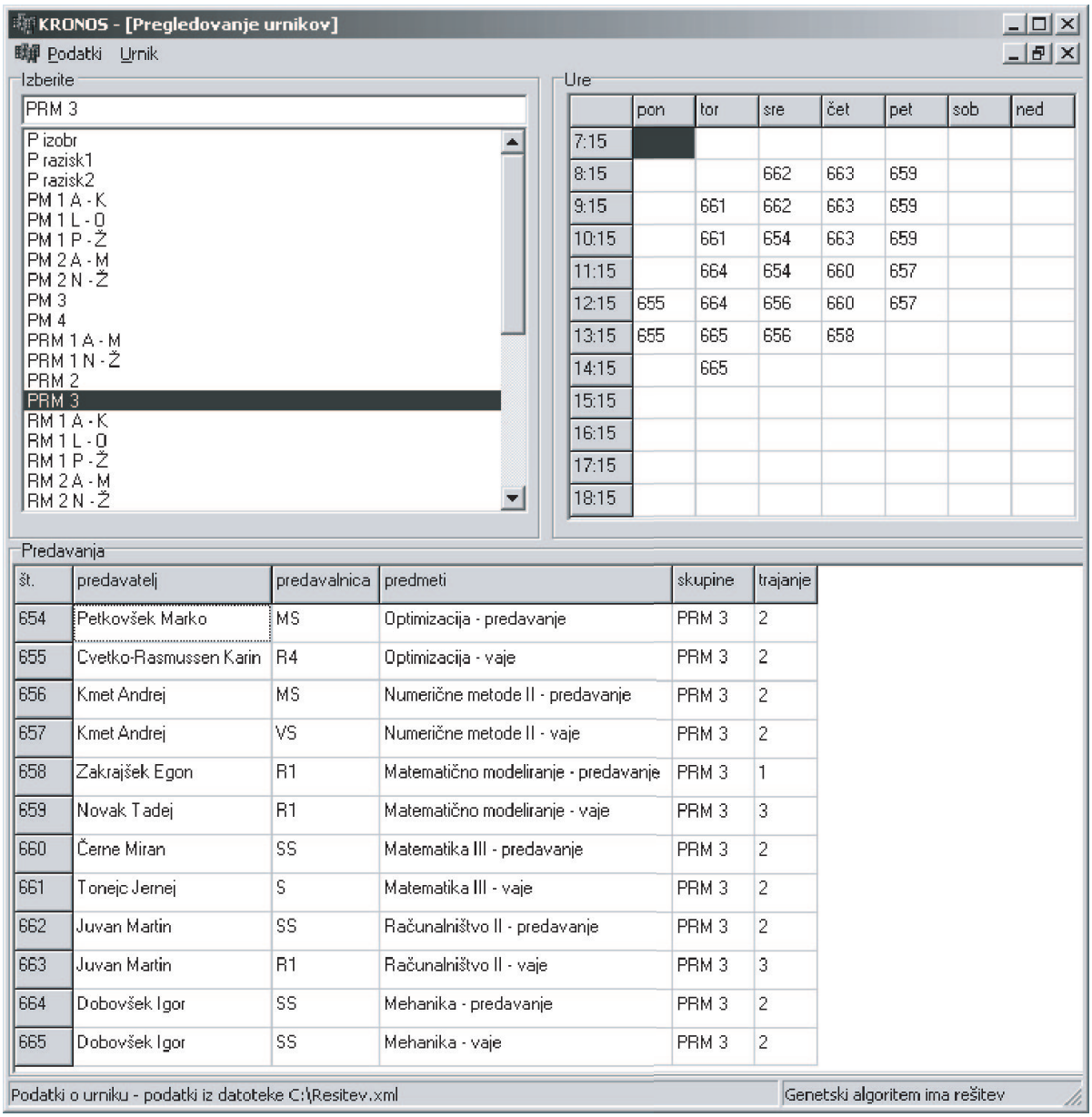
Ne glede na to, ali smo evolucijo ustavili sami ali je prišla do konca, nam program v podatkovne strukture shrani najboljši dobljeni urnik.



Slika 18: Program Kronos: generiranje urnikov.

5. Ko smo dobili rešitev za problem urnika, si jo lahko ogledamo. V meniju **Urnik → Pregledovanje urnikov** izberemo eno izmed možnosti **Po predavalnicah**, **Po predavateljih** ali **Po skupinah študentov**. Denimo, da smo izbrali pregledovanje po skupinah študentov. Odpre se okno, v katerem imamo seznam vseh

skupin študentov in dve tabeli (slika 19). Zgornja predstavlja urnik, v spodnji pa so zbrani podatki o predavanjih. Število, ki je napisano v zgornji tabeli, poiščemo v spodnji tabeli in tako vidimo podrobnosti izbranega predavanja. (Če pregledujemo datoteko, ki namesto rešitve vsebuje le podatke za problem urnika, vidimo le fiksna predavanja.)



Slika 19: Program Kronos: pregledovanje urnikov po skupinah študentov.

Podobno poteka pregledovanje urnikov po predavalnicah in po predavateljih.

6. Če želimo shraniti dobljeni urnik, v meniju izberemo **Podatki → Shrani podatke** in podatke shranimo v XML datoteko.

4.2 Parametri genetskega algoritma

V prejšnjih razdelkih smo že govorili o parametrih genetskega algoritma in o njihovem vplivu na kakovost rešitev. V tem razdelku bomo naše trditve podkrepili s podrobno analizo vpliva parametrov.

Vpliv parametrov na uspešnost genetskega algoritma bomo ponazorili tako, da bomo v začetni konfiguraciji parametrov spreminjali po en parameter. Tako bomo videli, kako ta parameter vpliva na kakovost rešitev.

4.2.1 Začetna konfiguracija

Za začetno konfiguracijo smo izbrali kombinacijo parametrov, ki v kratkem času vrne dober urnik. Izbrane vrednosti parametrov so podane v tabeli 8:

parameter	**vrednost**
število generacij	1000
število osebkov	20
odstotek osebkov v eliti	25 %
verjetnost mutacije	0.5 %
verjetnost pri prekrižanju	50 %
utež za razrede	0.01
utež za proste ure	0.2
utež za selitve	0.0001
kazen za selitev med stavbami	5
kazen za selitev med predavalnicami	3

Tabela 8: Vrednosti parametrov v začetni konfiguraciji.

Parameter verjetnost mutacije nam pove, kolikšna je verjetnost, da bomo uporabili mutacijo na posameznem predavanju osebka (glej stran 48). Parameter verjetnost pri prekrižanju pomeni razmerje genov, ki jih bo osebek dobil od staršev pri prekrižanju (glej stran 47). Uteži, s katerimi določimo pomembnost šibkih omejitev, smo opisali že na straneh 22–24. Z utežjo za razrede določimo upoštevanost zaželenih ur za razrede.

Utež za proste ure nam pove, koliko se bo upoštevala omejitev, ki zahteva čim manj prostih ur za študente. Podobno utež za selitve pove, koliko se bo upoštevala omejitev, naj se študenti čim manj selijo.

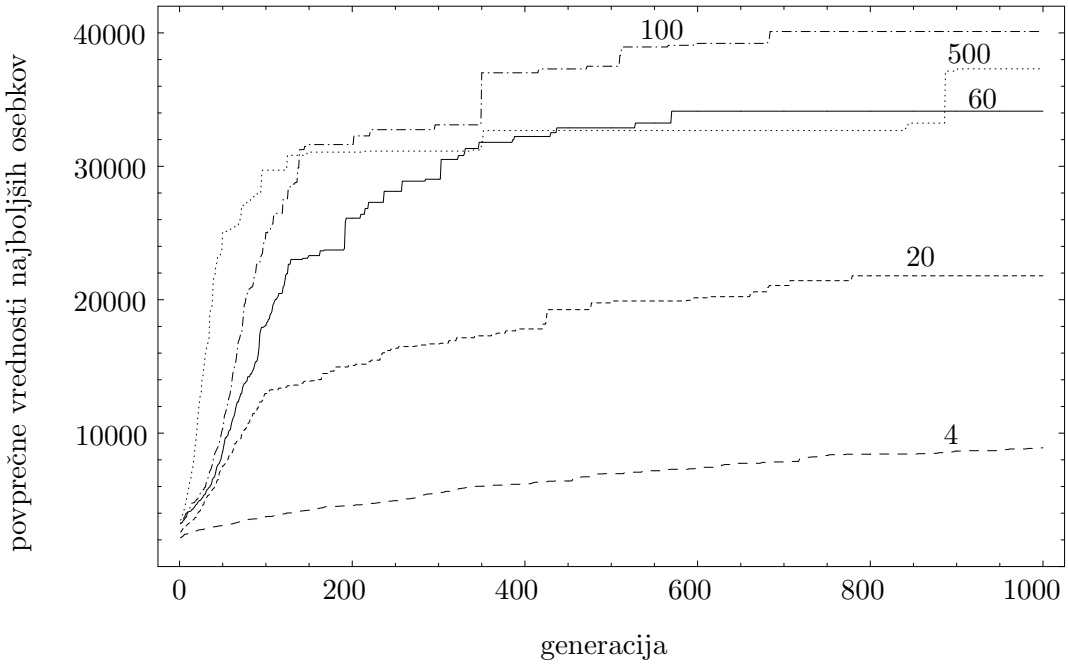
4.2.2 Ugotavljanje vpliva parametrov

Obnašanje programa smo preizkusili z naslednjimi variacijami začetne konfiguracije:

1. Število osebkov: 4, 20, 60, 100, 500.
2. Odstotek osebkov v eliti: 0, 5, 10, 25, 50, 80.
3. Verjetnost mutacije (v odstotkih): 0, 0.1, 0.3, 0.5, 0.7, 1, 5, 20.
4. Verjetnost pri prekrižanju (v odstotkih): 0, 5, 10, 20, 40, 50.

Za vsak nabor parametrov smo program pognali desetkrat. Pri vseh poskusih je bila prva generacija za vse nabore enaka (razen seveda tam, kjer je število osebkov različno). Na grafih, ki sledijo, so prikazane povprečne vrednosti najboljših urnikov v teh desetih poskusih.

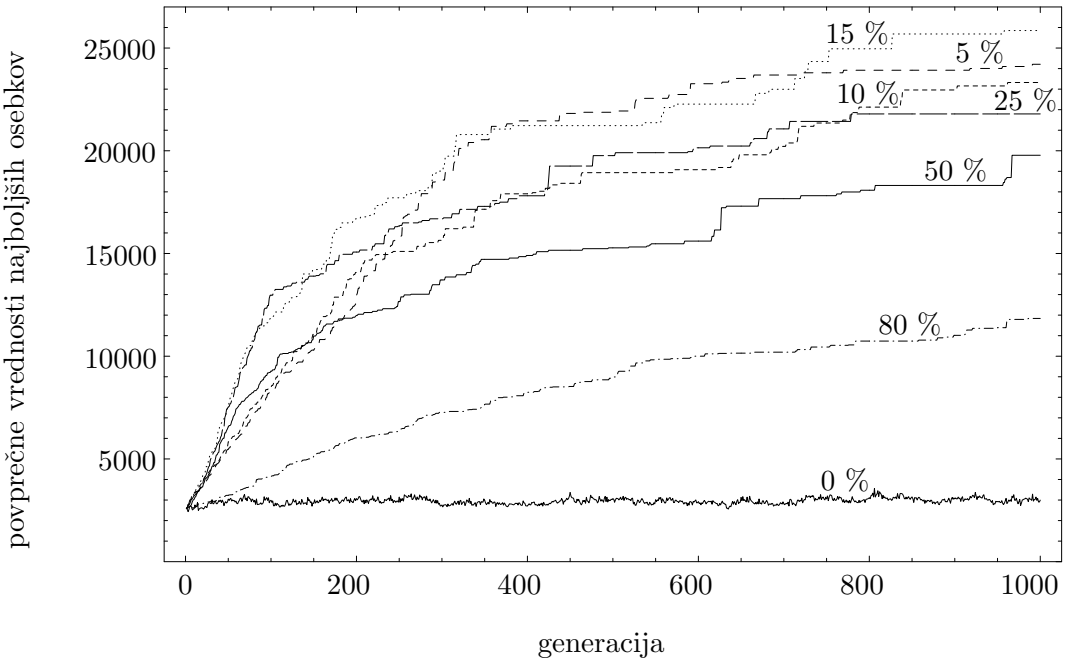
Število osebkov. Na sliki 20 so prikazane vrednosti najboljših urnikov pri različnem številu osebkov.



Slika 20: Preizkušanje parametra: število osebkov.

Takoj lahko vidimo, da so za uspešno evolucijo štirje osebki premajhna populacija in je izboljševanje vrednosti najboljše rešitve prepočasno. Tudi 20 osebkov nam še ne da dovolj dobrih rezultatov. Če želimo dober urnik, moramo v populaciji imeti vsaj več deset osebkov.

Odstotek osebkov v eliti. Na sliki 21 lahko vidimo, kako je kvaliteta urnika odvisna od odstotka elitnih osebkov v populaciji genetskega algoritma.

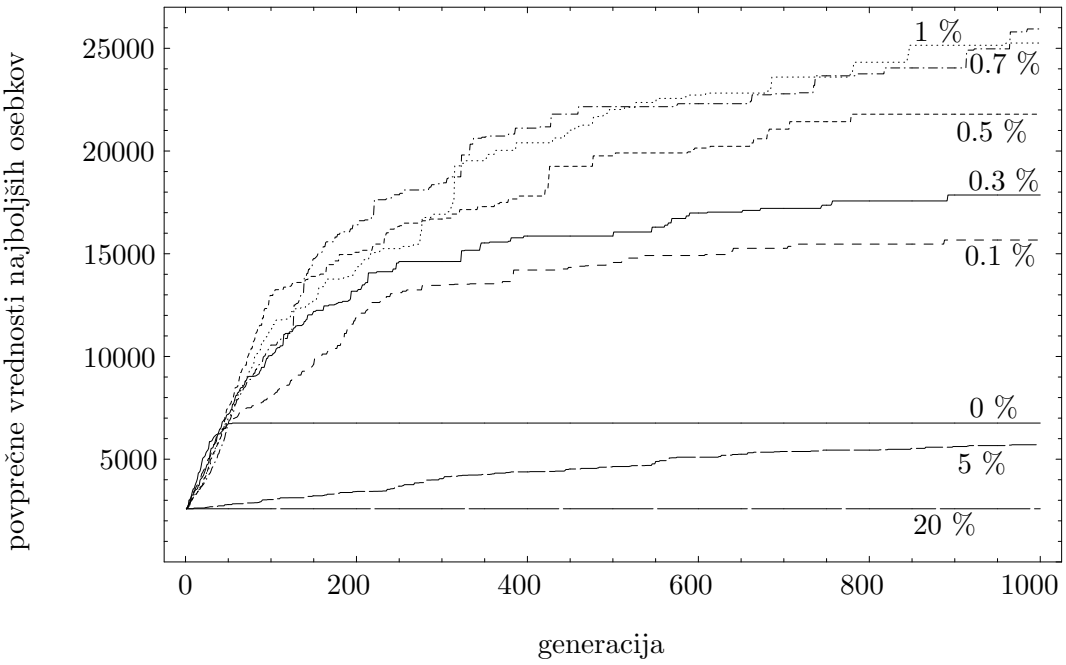


Slika 21: Preizkušanje parametra: odstotek osebkov v eliti.

Na sliki 21 najbolj izstopa žagasta funkcija, ki predstavlja vrednost najboljšega osebka ko nimamo elite. Takrat se namreč v vsakem koraku genetskega algoritma populacija v celoti zamenja. Vidimo, da v takem primeru genetski algoritem sploh ne dobi bistveno boljše rešitve od začetne, tudi če ga izvajamo zelo dolgo časa. Ta preizkus nam predvsem pokaže, da je elitizem pri reševanju problema urnika nujno potreben.

Sicer pa so rezultati, ki smo jih dobili pri tem preizkusu, pričakovani. Najboljša je ne prevelika elita, ki omogoča, da se nekaj najboljših osebkov ohranja, hkrati pa pusti dovolj prostora tudi za nove osebke. Prevelika elita (50 ali 80 odstotna) „zaduši" izboljševanje vrednosti najboljšega osebka, saj je novih osebkov na posameznem koraku premalo.

Verjetnost mutacije. Na sliki 22 lahko vidimo, kako se kvaliteta urnika spreminja glede na verjetnost mutacije.



Slika 22: Preizkušanje parametra: verjetnost mutacije.

Če izberemo verjetnost mutacije 0 % , to v praksi pomeni, da je prekrižanje edina metoda, s katero pridobivamo nove osebke. Kot lahko vidimo iz slike, se pomanjkanje mutacije občuti že po 50-ih generacijah, ko se ustavimo v lokalnem optimumu, iz katerega nas bi lahko premaknila le mutacija.

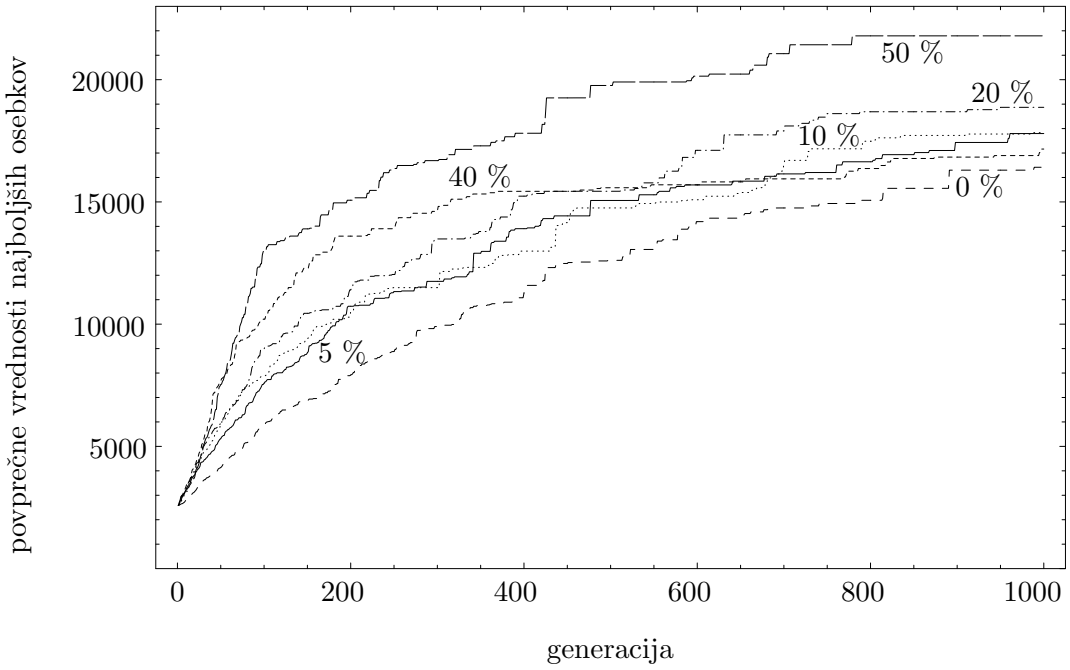
Po drugi strani pa prevelika verjetnost mutacije (20 in več odstotna) pridela same nekakovostne rešitve, ki ostajajo na ravni začetnih naključnih rešitev.

Pri testiranju parametrov na naši začetni konfiguraciji se izkaže, da je najboljša mutacija nekje med 0.5 in 1 %.

Verjetnost pri prekrižanju. Na sliki 23 lahko vidimo vrednosti najboljših urnikov pri različni verjetnosti pri prekrižanju.

Če je verjetnost pri prekrižanju enaka 0 %, to pomeni, da je potomec zelo podoben enemu izmed staršev. Edina razlika med njima je tista, ki jo povzroči mutacija. Evolucija s takšnim prekrižanjem je pričakovano najslabša, saj razvija posamezne osebke brez kombinacij med njimi.

Sicer pa lahko vidimo, da nam najboljše rezultate prinese 50-odstotno prekrižanje, ki najbolj „premeša” genetski material.



Slika 23: Preizkušanje parametra: verjetnost pri prekrižanju.

Pri opravljenem testiranju smo vsakič spreminjali le en parameter. Najboljše vrednosti parametrov, ki smo jih pri tem dobili, so bile najboljše le pri danih vrednostih ostalih parametrov. V splošnem torej ne velja, da je 15-odstotna elita najboljša za vse nabore ostalih parametrov. Tako nam nabor parametrov, ki ga sestavimo iz najboljših parametrov pri posameznih testiranjih, ne vrne najboljšega urnika.

Pri vseh preizkusih je bil uporabljen generator naključnih števil, ki je vgrajen v okolju Borland C++ Builder.

4.3 Rezultati

Program Kronos smo preizkusili na podatkih z Oddelka za matematiko Fakultete za matematiko in fiziko Univerze v Ljubljani za letni semester študijskega leta 2001/2002.

Imeli smo naslednje podatke:

- 84 ur (12 ur na dan, 7 dni na teden, perioda urnika 1 teden),
- 21 predavalnic v 5 stavbah,
- 35 skupin študentov v 21 razredih,
- 147 predmetov,

– 71 predavateljev,

– 355 srečanj v 134 skupinah srečanj in

– 161 predavanj, od tega 20 fiksnih.

Poleg tega smo imeli določenih 833 prepovedanih ur za predavalnice in 2014 prepovedanih ur za predavatelje. Vse predavalnice so imele prepovedane sobotne in nedeljske ure. Nekatere predavalnice (fizikalne predavalnice, ki jih na isti fakulteti uporablja predvsem Oddelek za fiziko) so bile na voljo le nekaj ur na teden. Pri predavateljih so bile prepovedane vse sobotne in nedeljske ure.

Ko smo iskali najboljši urnik za Oddelek za matematiko Fakultete za matematiko in fiziko, smo uporabili nekoliko drugačne parametre kot pri ugotavljanju vpliva posameznih parametrov. Želeli smo predvsem urnik, ki bi imel čim manj lukenj in v katerem bi večina predavanj potekala med 8. in 14. uro. Zato smo zahtevo po čim manj selitvah študentov zanemarili (utež za selitve smo postavili na 0).

Najboljši urnik je bil dobljen s parametri, ki so zbrani v tabeli 9:

parameter	**vrednost**
število generacij	1000
število osebkov	500
število osebkov v eliti	125
verjetnost mutacije	0.5 %
verjetnost pri prekrižanju	50 %
utež za razrede	0.01
utež za proste ure	0.2
utež za selitve	0

Tabela 9: Vrednosti parametrov pri najboljšem urniku.

Dobljeni urnik bomo prikazali tako, da bomo za vsak razred na isti sliki združili urnike vseh skupin tega razreda. Razredi so razdeljeni največ na 3 skupine. Najtemnejši deli na sliki predstavljajo tista predavanja, ki jih imajo hkrati vse tri skupine razreda, malo svetleje so pobarvane ure, ki jih imata dve skupini, najsvetlejši deli pa so ure predavanj le ene skupine študentov.

Zaradi preglednosti so na slikah izpuščene ure, ki so brez predavanj (sobote in nedelje, prva jutranja ura in nekatere popoldanske ure). Prav tako nismo narisali urnikov

podiplomskih programov, ker so bila vsa njihova predavanja fiksna in njihovi urniki zato niso dobljeni z genetskim algoritmom.

Prikazali bomo urnike za:

– visokošolski strokovni študij praktične matematike,

– univerzitetni študij matematike – uporabna smer,

– univerzitetni študij matematike – pedagoška smer,

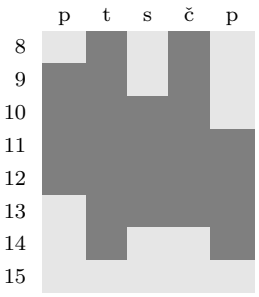
– univerzitetni študij matematike – teoretična smer,

– univerzitetni študij matematike – računalništvo z matematiko.

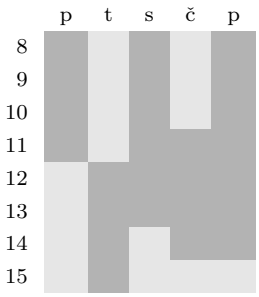
Smeri z manj študenti so v nižjih letnikih razdeljene na več skupin, da se pri predmetih omogoči delitev na skupine po abecedah. V višjih letnikih imajo študenti vseh smeri na voljo izbirne predmete. Ker ne želimo, da bi se ti predmeti prekrivali, smo v višjih letnikih iz posameznega razreda sestavili le eno skupino študentov.

Praktična matematika

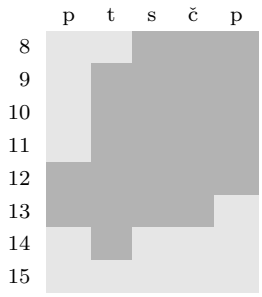
1. letnik (2 skupini)



2. letnik (1 skupina)

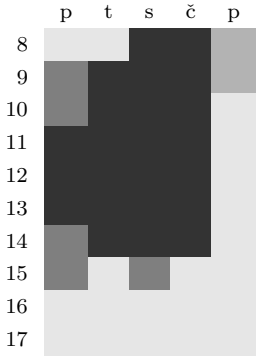


3. letnik (1 skupina)

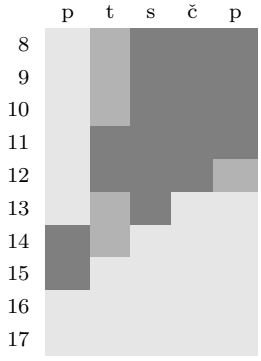


Uporabna smer

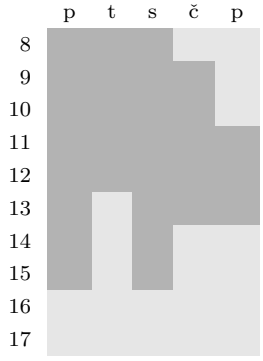
1. letnik (3 skupine)



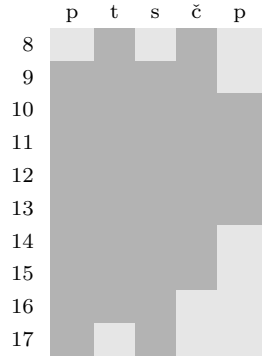
2. letnik (2 skupini)



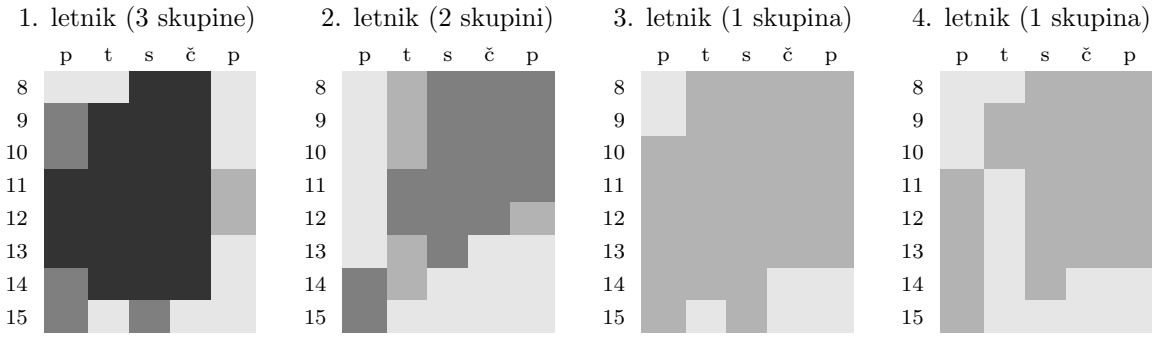
3. letnik (1 skupina)



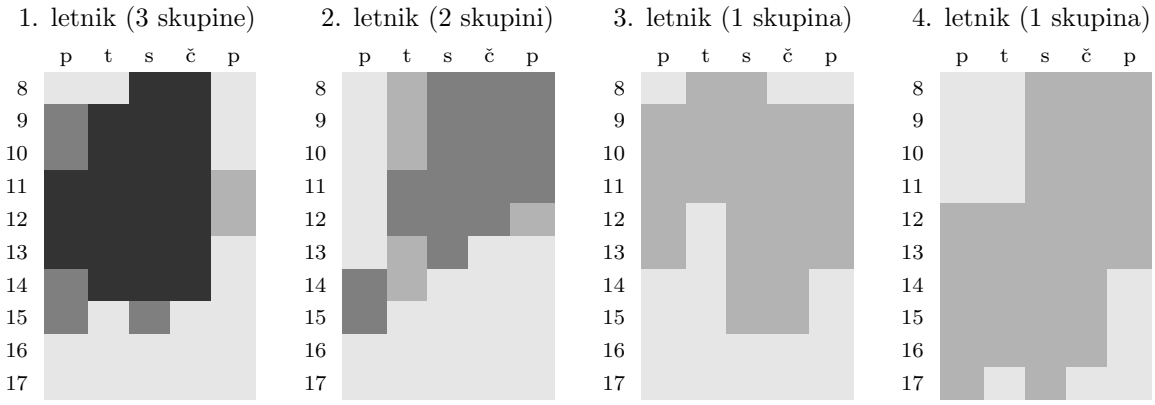
4. letnik (1 skupina)



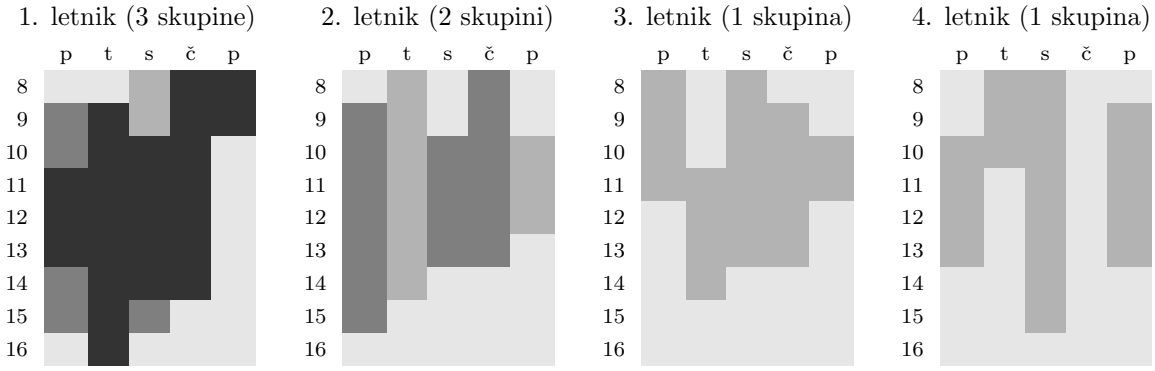
Pedagoška smer



Teoretična smer



Računalništvo z matematiko



Iz urnikov lahko vidimo, da je program naredil rešitev, ki upošteva najbolj zaželeni lastnosti urnika. To sta zahtevi, naj ima urnik za študente čim manj prostih ur in naj predavanja potekajo večinoma med 8. in 14. uro.

Program je za generiranje teh urnikov na računalniku s procesorjem AMD Athlon 1.2 GHz in pomnilnikom velikosti 512 MB porabil 55 minut.

5 Zaključek

V diplomskem delu smo si ogledali problem urnika in njegovo reševanje z genetskim algoritmom. Videli smo, da je genetski algoritem dobra metoda za reševanje tega problema, saj nam vrne urnik v skladu z našimi željami.

V programu Kronos smo za Fakulteto za matematiko in fiziko implementirali le tri šibke omejitve, po katerih ocenjujemo kakovost urnika. Zlahka bi k algoritmu dodali še kakšno šibko (ali pa tudi strogo) omejitev. Nekaj primerov:

- Predavanja naj bodo za skupine študentov razporejena enakomerno čez cel teden.
- V dnevih, ko ima skupina študentov predavanja od jutra do večera, naj ima prosto uro v času kosila.
- Kapaciteta predavalnic naj bo čim bolje izkoriščena.
- Fakulteta (ali šola) naj bo odprta čim manj časa (vsa predavanja naj se zvrstijo v čim krajšem času).

Poleg tega bi lahko bolj upoštevali želje predavateljev z naslednjo izboljšavo. Namesto vrednosti 0 in 1 za uro t, ki nam povesta, ali ima v uri t predavatelj čas, bi lahko tudi tu uvedli uteži: utež 0 za uro t bi še vedno pomenila zahtevo, da predavatelju v uri t ne smemo dodeliti predavanj, vsaka druga pozitivna utež pa bi pomenila zaželenost tiste ure. Iz te izboljšave bi tako poleg stroge omejitve naredili še šibko omejitev, ki bi poskusila čim bolj upoštevati zaželene ure predavateljev. Lahko pa bi k vsakemu predavatelju dodali še eno utež, ki bi določala „pomembnost" predavatelja. (Predavatelju z večjo utežjo bi se zaželene ure bolj upoštevale kot tistemu z manjšo utežjo.)

Podobno kot pri urah predavateljev bi lahko tudi iz zaželenih ur za razrede naredili strogo in šibko omejitev. Tudi tu bi ura z utežjo 0 pomenila prepovedano uro, ostale uteži pa bi predstavljale šibko omejitev. Ta način bi bil najbolj uporaben pri fakultetah, kjer se nekateri programi izvajajo med tednom, drugi pa le za vikend (redni in izredni programi).

Poleg naštetih izboljšav naš model problema urnika omogoča tudi večtedenske periode, ki pa jih v programu Kronos nismo implementirali.

Genetski algoritem bi lahko vgradili v aplikacijo, ki bi poleg izbire uteži šibkim omejitvam in avtomatskega generiranja urnika omogočala še:

- vnos in spremembo vhodnih podatkov,
- izhajanje iz že obstoječega urnika (npr. za prejšnje leto),
- ročno popravljanje avtomatsko dobljenega urnika in
- izpisovanje dobljenega urnika.

Z nadgradnjo, ki bi upoštevala navedene možnosti, bi program lahko ustrezal mnogim fakultetam in šolam.

Tudi sam genetski algoritem bi lahko še izboljšali. Možnost, ki bi prinesla dobre rezultate, je spreminjanje verjetnosti mutacije med evolucijo. Kot smo videli pri preizkušanju parametrov, se rešitve hitro izboljšujejo, dokler ne pridejo do nekega lokalnega optimuma. Potem pa lahko le s pomočjo mutacije „preskočijo" na drug (boljši) nivo. Takrat bi s povečanjem verjetnosti mutacije dosegli hitrejše preskoke v kakovosti urnika.

Čeprav je naš genetski algoritem zasnovan tako, da lahko z malo dodatnega dela vanj vključimo marsikatero šibko ali strogo omejitev, ima tudi on svoje meje. Pri definiciji urnika smo namreč privzeli, da predavanje predava le en predavatelj hkrati. Če bi želeli imeti hkrati več predavateljev za isto predavanje, bi morali izbrani model urnika spremeniti. Prav tako smo privzeli, da v enem predavanju skupina študentov lahko posluša le en predmet hkrati. Tudi to bi lahko spremenili, a bi pri tem morali poseči v definicijo problema urnika.

S tem smo prišli do vprašanja, kako natančen (oz. splošen) model potrebujemo, da bo zadovoljeval potrebe čim večjega števila fakultet ali šol. To ostaja odprt problem, ki je lahko delno rešljiv le v okviru ene države oz. območja s podobnim študijskim sistemom.

Na splošno menim, da so genetski algoritmi dobra metoda za reševanje problema urnika in bodo v prihodnosti (z raznimi izboljšavami) prinesli še boljše rezultate.

Literatura

[1] S. Abdennadher in M. Marte. University course timetabling using constraint handling rules. *Applied Artificial Intelligence 14*, str. 311–325, 2000.

[2] V. A. Bardadym. Computer-aided school and university timetabling: The new wave. V Burke in Ross [8], str. 22–43.

[3] E. K. Burke in M. W. Carter, urednika. *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT'97, Toronto, Canada, August 20-22, 1997, Selected Papers*, *Lecture Notes in Computer Science*, zv. 1408. Springer, 1998.

[4] E. K. Burke, D. G. Elliman, P. H. Ford in R. F. Weare. *Specialised Recombinative Operators for the Timetabling Problem*, str. 75–85. Springer-Verlag, 1995.

[5] E. K. Burke, D. G. Elliman in R. F. Weare. A genetic algorithm based university timetabling system. 1994.

[6] E. K. Burke in W. Erben, urednika. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16-18, 2000, Selected Papers*, *Lecture Notes in Computer Science*, zv. 2079. Springer, 2001.

[7] E. K. Burke, J. P. Newall in R. F. Weare. A memetic algorithm for university exam timetabling. V Burke in Ross [8], str. 241–250.

[8] E. K. Burke in P. Ross, urednika. *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, *Lecture Notes in Computer Science*, zv. 1153. Springer, 1996.

[9] M. P. Carrasco in M. V. Pato. A multiobjective genetic algorithm for the class/teacher timetabling problem. V Burke in Erben [6], str. 3–17.

[10] M. Carter in G. Laporte. Recent developments in practical course timetabling. V Burke in Carter [3], str. 3–19.

[11] A. Colorni, M. Dorigo in V. Maniezzo. A genetic algorithm to solve the timetable problem. Tehnično poročilo 90-060, 1990. Politecnico di Milano, Italija.

[12] A. Colorni, M. Dorigo in V. Maniezzo. Genetic algorithms and highly constrained problems: the time-table case. V H.-P. Schwefel in R. Manner, urednika, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, (PPSN) 1*, zv. 496, str. 55–59, Dortmund, Nemčija, 1991. Springer-Verlag, Berlin, Nemčija.

[13] D. Costa. A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research 76*, str. 98–110, 1994.

[14] L. Curk in G. Budimir. XML – nov jezik na svetovnem spletu. *COBISS obvestila*, letnik 4, zv. 2, str. 1–14, 1999.

[15] S. Deris, S. Omatu, H. Ohta in P. Saad. Incorporating constraint propagation in genetic algorithm for university timetable planning. *Engineering Applications of Artificial Intelligence 12*, str. 241–253, 1999.

[16] K. A. Dowsland. Off-the-peg or made-to-measure: Timetabling and scheduling with simulated annealing and tabu search. V Burke in Carter [3], str. 37–43.

[17] S. Elmohamed, P. Coddington in G. Fox. A comparison of annealing techniques for academic course scheduling. V Burke in Carter [3], str. 92–112.

[18] J. Čer-Titan. Urnik kot optimizacijski problem. Magistrska naloga, Univerza v Mariboru, Pedagoška fakulteta, 1995.

[19] S. Even, A. Itai in A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing 5*, str. 691–703, 1976.

[20] C. Gueret, N. Jussien, P. Boizumault in C. Prins. Building university timetables using constraint logic programming. V Burke in Ross [8], str. 130–145.

[21] S. Gyori, Z. Petres in A. R. Varkonyi-Koczy. Genetic algorithms in timetabling. A new approach, rokopis, 2001 (http://www.mft.hu/hallg/200107.pdf).

[22] B. M. Kelley. Genetic programming and a genetic algorithm for approximate optimal timetable design, rokopis, 2001 (http://www.cs.csubak.edu/bkelley/presentations/csc540/540P2.doc).

[23] B. M. Kelley. Timetable is np-complete, rokopis, 2001 (http://www.cs.csubak.edu/bkelley/presentations/csc540/540P1.doc).

[24] J. Leskovec. Clp. (constraint logic programming). http://ai.ijs.si/jure/mat/clp-98.html.

[25] M. Mitchell. *An Introduction to Genetic Algorithms.* A Bradford Book, The MIT Press, 1996.

[26] D. C. Rich. A smart genetic algorithm for university timetabling. V Burke in Ross [8], str. 181–196.

[27] H. Ueda, D. Ouchi, K. Takahashi in T. Miyahara. A co-envolving timeslot/room assignment genetic algorithm technique for university timetabling. V Burke in Erben [6], str. 48–63.

[28] R. F. Weare, E. K. Burke in D. G. Elliman. A hybrid genetic algorithm for highly constrained timetabling problems. 1995.

[29] G. M. White in J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. V Burke in Carter [3], str. 187–198.