SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2854

RJEŠAVANJE PROBLEMA RASPOREĐIVANJA MEDICINSKOG OSOBLJA KORIŠTENJEM METAHEURISTIKA

Ivan Navratil

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2854

RJEŠAVANJE PROBLEMA RASPOREĐIVANJA MEDICINSKOG OSOBLJA KORIŠTENJEM METAHEURISTIKA

Ivan Navratil

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2854

Pristupnik: **Ivan Navratil (0036510118)**

Studij: Računarstvo

Profil: Računarska znanost

Mentor: doc. dr. sc. Marko Đurasević

Zadatak: **Rješavanje problema raspoređivanja medicinskog osoblja korištenjem metaheuristika**

Opis zadatka:

Proučiti problem raspoređivanja medicinskog osoblja u bolnicama. Istražiti heurističke i metaheurističke postupke koji su korištenu u literature za rješavanje razmatranog problema. Pronaći odgovarajuće primjerke problema koji će biti korišteni za ispitivanje razvijenih postupaka. Odabrati i implementirati prikladni metaheuristički postupak i prilagoditi ga za razmatrani problem. Ocijeniti efikasnost implementiranog algoritma na odabranim primjercima problema i predložiti moguća poboljšanja s ciljem poboljšanja rezultata. Usporediti ostvarene rezultate s postojećim rezultatima. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 27. lipnja 2022.

SADRŽAJ

1. Uvod **1**

2. O problemu raspoređivanja medicinskog osoblja **2**

3. GRASP (Greedy randomized adaptive search procedure) **3**

3.1. Faza izgradnje rješenja 3

3.2. Faza lokalne pretrage rješenja 4

4. Variable Neighborhood Descent (VND) **6**

5. Model i implementacija rješenja **8**

5.1. Tvrda ograničenja 8

5.1.1. Ograničenje jedne smjene dnevno 8

5.1.2. Ograničenje iduće smjene 8

5.1.3. Maksimalan broj određene smjene po radniku 9

5.1.4. Minimalni i maksimalni broj radnih minuta 10

5.1.5. Minimalni i maksimalni broj uzastopnih smjena 10

5.1.6. Minimalni broj uzastopnih slobodnih dana 11

5.1.7. Maksimalni broj radnih vikenda 12

5.1.8. Slobodni dani 12

5.2. Meka ograničenja 13

5.2.1. Zahtjev za smjenom i zahtjev protiv smjene 13

5.2.2. Pokriće smjena 14

5.3. GRASP - konstrukcijska faza 14

5.3.1. Podjela poslova 15

5.3.2. Problemi zadovoljavanja tvrdih ograničenja 15

5.3.3. Način pretrage 16

5.3.4. Spajanje rješenja u blokove 16

5.3.5. Odabir rješenja 18

5.3.6. Primjer stanja tijekom pretraživanja 19
5.4. GRASP - faza lokalne pretrage / VND 20
5.4.1. Prvi algoritam lokalne pretrage 21
5.4.2. Drugi algoritam lokalne pretrage 23
5.4.3. Vertikalna zamjena 25
5.4.4. Horizontalna zamjena 26
5.5. Konačni algoritam 26

6. Konfiguracija programskog rješenja **28**
6.1. Primjer konfiguracije programskog rješenja 28
6.2. Opis konfiguracije programskog rješenja 29

7. Usporedba i moguće nadogradnje rješenja **32**

8. Zaključak **36**

Literatura **37**

Popis slika **39**

Popis tablica **40**

A. Format ulazne datoteke **43**

1. Uvod

Problem izrade optimalnih smjena radnika interesantan je problem jer je kompleksan za rješavanje, a potrebno ga je periodično izraditi u gotovo svim smjenskim poslovima. Izrada optimalnog ili gotovo optimalnog rasporeda bitna je za zadovoljstvo radnika, ali je također bitna za poslodavce kako bi minimizirali svoje troškove.

Problem izrade optimalnih smjena radnika promatra se više od 50 godina te se tijekom vremena pristup problemu mijenjao pa su neke tehnike prevladale nad drugima. Na početku, većina metoda rješavanja problema temeljila se na matematičkom pristupu problemu gdje bi se određena ciljna funkcija optimirala prema nekim ograničenjima. Znanstvenici su pokušali razviti linearne modele problema. Budući da je problem veoma težak za rješavanje, u situacijama u stvarnom životu, pronalazak optimalnog rješenja je vremenski neisplativ, a često i besmislen. U stvarnim situacijama je bitno da se izradi visoko kvalitetan raspored u razumnom vremenu gdje su zadovoljeni svi tvrdi i što više mekih uvjeta umjesto da se pronađe optimalno rješenje za čiji izračun bi bilo potrebno značajno više vremena.

U ovom radu iskorišteni su metaheuristički algoritmi GRASP (Greedy randomized adaptive search procedure) i VND (Variable neighborhood descent) za izradu rasporeda radnika.

2. O problemu raspoređivanja medicinskog osoblja

Problem raspoređivanja medicinskih sestara naziv je za problem u kojem je medicinskim sestrama potrebno dodijeliti smjene prema njihovim zahtjevima, uzevši u obzir niz drugih uvjeta. Rezultat izrade rasporeda je tablica dimenzija $n \times m$ gdje je n broj medicinskih sestara, m period planiranja, a vrijednost ćelije smjena koju određeni radnik radi određeni dan.

Verzija problema koja se rješava opisana je u [3]. Definirana su tvrda i meka ograničenja problema.

Tvrda ograničenja problema su ograničenje jedne smjene dnevno, ograničenje iduće smjene, maksimalan broj određene smjene po radniku, minimalni i maksimalni broj radnih minuta, minimalni i maksimalni broj uzastopnih smjena, minimalni broj uzastopnih slobodnih dana, maksimalni broj radnih vikenda te slobodni dani radnika. Meka ograničenja problema su zahtjeva za smjenom, zahtjev protiv smjene te pokriće smjene.

Sva tvrda ograničenja moraju biti zadovoljena. Za svako prekršeno meko ograničenje određena je kazna. Kršenje ograničenja zahtjeva za smjenom ili zahtjeva protiv smjene, u pravilu, imaju kazne od nekoliko bodova. Kršenje pokrića smjene s prevelikim brojem radnika, u pravilu, ima kaznu od nekoliko bodova po radniku dok kršenje pokrića smjene s premalim brojem radnika ima kaznu od 100 bodova po radniku.

Cilj problema je pronaći raspored koji zadovoljava sva tvrda ograničenja te ima što manju kaznu koja je prouzročena kršenjem mekih ograničenja.

3. GRASP (Greedy randomized adaptive search procedure)

Algoritam GRASP je metaheuristički algoritam često korišten u optimizaciji kombinatoričkih problema. Svaka iteracija GRASP metode je nezavisna, ne održava se memorija pretraživanih stanja. Jedini podatak koji se prenosi iz iteracije u iteraciju je rješenje koje je odabrano u prethodnoj iteraciji ili prazno rješenje u slučaju prve iteracije.

Svaka iteracija GRASP metode sastoji se od dvije faze - faze izgradnje rješenja i faze lokalnog pretraživanja rješenja. U isječku 1 dan je pseudokod algoritma GRASP čiji je cilj minimizirati vrijednost kazne.

```
1  fun GRASP() {
2    do{
3      var solution = ConstructSolution()
4      solution = LocalSearch(solution)
5      if(solution.penalty <  best.penalty){
6        best = solution
7      }
8    } while(stoppingCondition.notMet())
9      return best
10 }
```

Isječak 1: Pseudokod algoritma GRASP.

3.1. Faza izgradnje rješenja

U fazi izgradnje rješenja, postupno se gradi prihvatljivo rješenje tj. rješenje koje zadovoljava sva tvrda ograničenja. Rješenje se gradi uporabom nasumičnog pohlepnog algoritma. Ako dobiveno rješenje nije prihvatljivo, vrše se procedure popravljanja ili

se stvara novo rješenje.

Nasumični pohlepni algoritam koristi iste principe kao i običan pohlepni algoritam, ali u postupku izvođenja algoritma ugrađena je nasumičnost kako bi se prilikom više izvođenja algoritma dobila različita rješenja. Umjesto da se u svakom koraku odabire rješenje koje nudi najveći profit ili najmanju kaznu, u nasumičnom pohlepnom algoritmu odabire se jedno od najboljih rješenja, ali ne nužno i najbolje rješenje.

U svakoj iteraciji, skup svih rješenja koji mogu biti uključeni u parcijalno rješenje sortiran je prema svojoj dobroti. Tada se iz tog skupa elemenata uzimaju najbolji elementi te se s njima izgradi nova lista - ograničena lista kandidata (eng. *Restricted Candidate List - RCL*). Element koji će biti uključen u parcijalno rješenje nasumično je odabran od svih kandidata koji se nalaze u ograničenoj listi kandidata. Tada se parcijalno rješenje ažurira s odabranim kandidatom te je završena jedna iteracija faze izgradnje algoritma.

Veličina ograničene liste kandidata ograničava se brojem elemenata ili kvalitetom rješenja u odnosu na najbolje rješenje.

Ako se veličina ograničene liste kandidata ograničava s brojem elemenata tada se iz skupa rješenja koji mogu biti uključeni u parcijalno rješenje uzima n najboljih rješenja.

Ako se veličina ograničene liste kandidata ograničava u odnosu na kvalitetu rješenja tada se iz skupa rješenja koji mogu biti uključeni u parcijalno rješenje uzimaju svi elementi koji su bolji od neke granice. Uz pretpostavku da se radi o minimizacijskom problemu, granica je dana izrazom 3.1.

$$c^{min} + \alpha(c^{max} - c^{min}) \tag{3.1}$$

U izrazu 3.1, c^{min} je rješenje s najmanjom vrijednošću kazne, a c^{max} je rješenje s najvećom vrijednošću kazne. Parametar α je parametar algoritma čija je vrijednost u intervalu $[0, 1]$.

Za vrijednosti parametra α koje su bliže 0, algoritam će biti pohlepniji tj. u ograničenu listu kandidata ući će rješenja koja su blizu najboljem rješenju. Za vrijednosti parametra α koje su bliže 1, granica za ulazak u ograničenu listu kandidata bit će viša, a kao posljedica toga nasumičnim odabirom iz ograničene liste kandidata nekada će se odabrati i rješenja znatno lošija od najboljeg rješenja u ograničenoj listi kandidata.

3.2. Faza lokalne pretrage rješenja

Rješenje dobiveno fazom izgradnje rješenja nije nužno lokalno optimalno. Kako bi se dobiveno rješenje poboljšalo, potrebno je izvesti postupke lokalne pretrage.

Tradicionalno, koriste se jednostavni postupci lokalnih pretraga, ali moguće je koristiti i druge metaheuristike koje koriste jedno rješenje (npr. tabu pretraga, simulirano kaljenje...).

Metode lokalne pretrage su specifične za rješavani problem. Često se koristi više postupaka lokalne pretrage koji poboljšavaju različite dijelove rješenja, a onda se lokalne pretrage mogu različito kombinirati, npr. moguće je stalno istim poretkom pokretati postupke lokalne pretrage, nasumično birati postupke lokalne pretrage ili postupke lokalne pretrage pokretati istim redoslijedom, ali ako jedan od postupka lokalne pretrage pronađe bolje rješenje onda ponovno pokrenuti sve postupke lokalne pretrage, krenuvši od prvog postupka lokalne pretrage.

```
1 fun localSearch(solution:SolutionBuilder){
2 while(solution.isNotLocallyOptimal())
3     solution.localSearch();
4 }
5 return solution
```

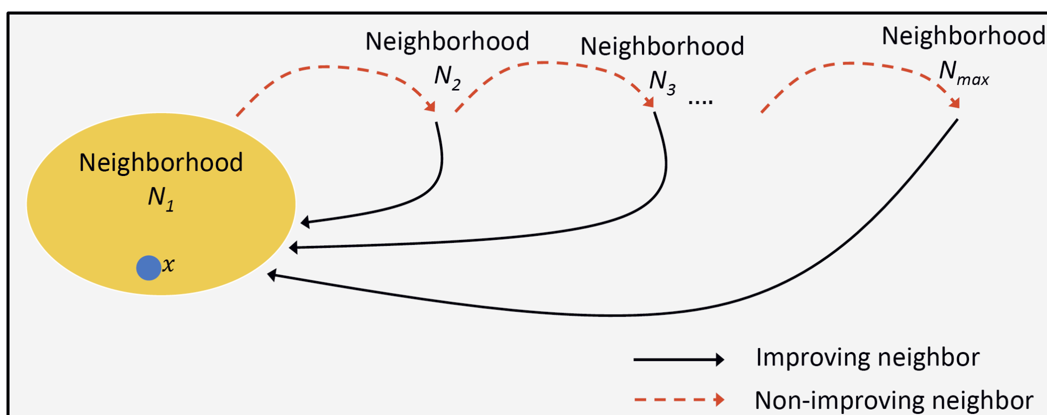
Isječak 2: Pseudokod lokalne pretrage rješenja.

4. Variable Neighborhood Descent (VND)

U fazi lokalne pretrage algoritma GRASP, upotrijebljen je algoritam VND. Algoritam uzastopno istražuje predefinirana susjedstva kako bi se pronašlo bolje rješenje.

VND pretražuje susjedstva determinističkim načinom. Redoslijed kojim su susjedstva istražena je proizvoljan, ali najčešća strategija je poredak susjedstva prema njihovoj složenosti, krenuvši od susjedstva najmanje složenosti prema susjedstvu najveće složenosti. Složenost susjedstva može biti definirana brojem susjeda, složenosti pronalaska susjeda ili nekom drugom mjerom.

Susjedstva su na slici 4.1 označena s $N_l, l = 1, ..., l_{max}$. Kao što je vidljivo na slici, krenuvši od nekog početnog rješenja istražuju se susjedstva redom, te ako se pronađe rješenje bolje od trenutnog rješenja tada se postupak pretraživanja ponavlja s novim rješenjem, a ako se ne pronađe bolje rješenje u niti jednom susjedstvu tada je pretraživanje završilo.



Slika 4.1: Ilustracija postupka pretraživanja algoritmom VND, izvor ilustracije: [8].

Pseudokod algoritma dan je u isječku 3.

```
1  fun variableNeighborhoodDescent(SolutionBuilder solution,
   ↪  List<Neighborhood> neighborhoods){
2
3  SolutionBuilder bestSolution = solution.copy()
4
5  for(int i = 0; i < neighborhoods.size ; i++){
6      SolutionBuilder newSolution =
       ↪  neighborhoods.get(i).search(solution);
7
8      if(newSolution.totalPenalty < solution.totalPenalty){
9          bestSolution = newSolution;
10         i = 0;
11 }
12 return bestSolution
```

Isječak 3: Pseudokod VND algoritma.

5. Model i implementacija rješenja

Primjer ulazne datoteke u kojoj su navedena ograničenja dan je u dodatku A.

5.1. Tvrda ograničenja

5.1.1. Ograničenje jedne smjene dnevno

Radniku ne može biti dodijeljena više od jedna smjena dnevno. Sve smjene traju kraće od 24 sata.

Ograničenje je inherentno zadovoljeno modelom programskog rješenja budući da je raspored radnika modeliran s `Map<String, Array<String?>>` gdje je ključ mape identifikacijska oznaka radnika, a vrijednost mape polje dužine jednake broju dana za koji se izrađuje raspored. Slobodan dan predstavljen je s `null` vrijednošću.

5.1.2. Ograničenje iduće smjene

U formulaciji problema zadano je da neke smjene ne smiju neposredno slijediti druge smjene te je takvo ograničenje navedeno u ulaznoj datoteci. Primjer navedenog ograničenja dan je u isječku 4. Vrijedi pretpostavka da su zadnji dan prethodnog perioda planiranja i prvi dan idućeg perioda planiranja slobodni dani.

Također, ovdje je određeno da smjene tipa `E` i smjene tipa `L` traju 480 minuta te se preko tih informacija vrši izračun zadovoljava li određeni radnik minimalni i maksimalni broj minuta koji treba odraditi.

```
SECTION_SHIFTS
# ShiftID, Length in mins, Shifts which cannot follow this shift |
↪  separated
E,480,
L,480,E
```

Isječak 4: Isječak ulazne datoteke u kojoj je navedeno ograničenje da ako radnik `i-ti` dan radi smjenu `L`, `i+1` dan ne može raditi smjenu `E`.

U programskom rješenju, kao što je vidljivo u isječku 5, ograničenje je modelirano s `FollowingShiftConstraint` razredom. U slučaju kršenja tog ograničenja, stvara se instanca razreda u kojoj je zabilježeno kojem radniku koji dan je prekršeno navedeno ograničenje.

```
data class FollowingShiftConstraint(val employeeId: String, val
↪  firstDay: Int) : HardConstraint
```

Isječak 5: Isječak programskog modela koji modelira ograničenje da neke smjene ne smiju slijediti druge smjene.

5.1.3. Maksimalan broj određene smjene po radniku

U formulaciji problema zadana je vrijednost koliko puta određeni radnik može raditi određenu smjenu. Primjer navedenog ograničenja dan je u isječku 6. U tom isječku, radnik s identifikacijskom oznakom `A` može raditi smjenu `E` maksimalno 14 puta te može raditi smjenu `L` maksimalno 14 puta.

```
# ID, MaxShifts, MaxTotalMinutes, MinTotalMinutes,
↪  MaxConsecutiveShifts, MinConsecutiveShifts,
↪  MinConsecutiveDaysOff, MaxWeekends
A,E=14|L=14,4320,3360,5,2,2,1
```

Isječak 6: Isječak ulazne datoteke u kojoj su navedena ograničenja za radnika s identifikacijskom oznakom `A`.

U programskom rješenju, kao što je vidljivo u isječku 7, ograničenje je modelirano s `MaxShiftsConstraint` razredom. U slučaju kršenja tog ograničenja, stvara se instanca razreda u kojoj je zabilježeno kojem radniku je koliko puta dodijeljena

određena smjena, a koje je ograničenje. Mora vrijediti $constraint < actual$ da bi se prekršilo ograničenje.

```
1 data class MaxShiftsConstraint(val employeeId: String, val shiftId:
  ↪ String, val constraint: Int, val actual: Int) :
2     HardConstraint
```

Isječak 7: Isječak programskog modela koji modelira maksimalan broj puta koji radnik može raditi određenu smjenu.

5.1.4. Minimalni i maksimalni broj radnih minuta

U formulaciji problema zadano je ograničenje minimalnog i maksimalnog broja minuta koje radnik mora odraditi u periodu planiranja. Ukupno vrijeme radnika mora biti između tih ograničenja. Primjer navedenog ograničenja dan je u isječku 6. U tom isječku, radnik s identifikacijskom oznakom `A` može raditi maksimalno 4320 minuta, a minimalno 3360 minuta.

U programskom rješenju, kao što je vidljivo u isječku 8, ograničenje je modelirano s `MinTotalMinutesConstraint` i `MaxTotalMinutesConstraint` razredima. U slučaju kršenja nekog od ograničenja, stvara se instanca razreda u kojoj je zabilježeno koji radnik radi ispod ili iznad danih ograničenja.

```
1 data class MinTotalMinutesConstraint(val employeeId: String, val
  ↪ constraint: Int, val actual: Int) : HardConstraint
2 data class MaxTotalMinutesConstraint(val employeeId: String, val
  ↪ constraint: Int, val actual: Int) : HardConstraint
```

Isječak 8: Isječak programskog modela koji modelira ograničenje minimalnog i maksimalnog broja minuta radnika.

5.1.5. Minimalni i maksimalni broj uzastopnih smjena

U formulaciji problema zadano je ograničenje minimalnog i maksimalnog broja uzastopnih smjena. Primjer navedenog ograničenja dan je u isječku 6. U tom isječku, radnik s identifikacijskom oznakom `A` mora raditi minimalno dva uzastopna dana, a može raditi maksimalno 5 uzastopnih dana.

Za minimalni broj uzastopnih smjena vrijedi pretpostavka da postoji beskonačni broj uzastopnih smjena na kraju prethodnog perioda planiranja i na početku idućeg

perioda planiranja. Drugim riječima, prvi radni blok, ako kreće od nultog dana, i zadnji radni blok, ako završava zadnjeg dana, mogu biti kraći od minimalnog broja uzastopnih smjena.

Za maksimalni broj uzastopnih smjena vrijedi pretpostavka da je posljednji dan prethodnog perioda planiranja bio slobodan te da je prvi dan idućeg perioda planiranja slobodan dan.

U programskom rješenju, kao što je vidljivo u isječku 9, ograničenje je modelirano s `MinConsecutiveShiftsConstraint` i `MaxConsecutiveShiftsConstraint` razredima. U slučaju kršenja nekog od ograničenja, stvara se instanca razreda u kojoj je zabilježeno koji radnik radi ispod ili iznad danih ograničenja te između kojih dana je prekršio ograničenje.

```
1 data class MinConsecutiveShiftsConstraint(val employeeId: String,
  ↪ val constraint: Int, val days: IntRange) :HardConstraint
2 data class MaxConsecutiveShiftsConstraint(val employeeId: String,
  ↪ val constraint: Int, val days: IntRange) :HardConstraint
```

Isječak 9: Isječak programskog modela koji modelira ograničenje minimalnog i maksimalnog broja uzastopnih smjena.

5.1.6. Minimalni broj uzastopnih slobodnih dana

U formulaciji problema zadan je minimalni broj uzastopnih slobodnih dana. Primjer navedenog ograničenja dan je u isječku 6. U tom isječku, radnik s identifikacijskom oznakom `A` mora imati minimalno 2 uzastopna slobodna dana.

Za ovo ograničenje vrijedi pretpostavka da postoji beskonačno uzastopnih slobodnih dana prije kraja prošlog perioda planiranja i na početku idućeg perioda planiranja.

U programskom rješenju, kao što je vidljivo u isječku 10, ograničenje je modelirano s `MinConsecutiveDaysOffConstraint` razredom. U slučaju kršenja nekog od ograničenja, stvara se instanca razreda u kojoj je zabilježeno kojem radniku je dodijeljen manji broj uzastopnih slobodnih dana od minimuma.

```
1 data class MinConsecutiveDaysOffConstraint(val employeeId: String,
  ↪ val constraint: Int, val days: IntRange) : HardConstraint
```

Isječak 10: Isječak programskog modela koji modelira ograničenje minimalnog broja uzastopnih slobodnih dana.

5.1.7. Maksimalni broj radnih vikenda

U formulaciji problema zadan je maksimalan broj radnik vikenda. Za radnika, vikend je radni ako radi u subotu ili u nedjelju. Primjer navedenog ograničenja dan je u isječku 6. U tom isječku, radnik s identifikacijskom oznakom `A` može imati maksimalno jedan radni vikend.

U programskom rješenju, kao što je vidljivo u isječku 11, ograničenje je modelirano s `MaxWeekendsConstraint` razredom. U slučaju kršenja broja radnih vikenda, stvara se instanca razreda u kojoj je zabilježeno kojem radniku je dodijeljeno više radnih vikenda od ograničenja.

```
1 data class MaxWeekendsConstraint(val employeeId: String, val
  ↪ constraint: Int, val actual: Int) : HardConstraint
```

Isječak 11: Isječak programskog modela koji modelira ograničenje maksimalnog broja radnih vikenda.

5.1.8. Slobodni dani

U formulaciji problema zadani su slobodni dani radnika. Primjer navedenog ograničenja dan je u isječku 13. U tom isječku, radnik s identifikacijskom oznakom `A` ne smije raditi treći dan (dani su indeksirani od nule).

```
1 SECTION_DAYS_OFF
2 # EmployeeID, DayIndexes (start at zero)
3 A,3
```

Isječak 12: Isječak ulazne datoteke u kojoj su određeni slobodni dani radnika.

U programskom rješenju, kao što je vidljivo u isječku 13, ograničenje je modelirano s `DayOffConstraint` razredom. U slučaju kršenja tog ograničenja, stvara se instanca razreda u kojoj je zabilježeno kojem radniku koji dan je prekršeno ograničenje.

```
1 data class DayOffConstraint(val employeeId: String, val day: Int) :
  ↪ HardConstraint
```

Isječak 13: Isječak programskog modela koji modelira ograničenje slobodnih dana za radnika.

5.2. Meka ograničenja

5.2.1. Zahtjev za smjenom i zahtjev protiv smjene

U formulaciji problema zadani su zahtjevi radnika za smjenom te zahtjevi radnika protiv smjene. Primjer navedenog ograničenja dan je u isječku 14. U tom isječku, radnik s identifikacijskom oznakom A dao je zahtjev da peti dan radi smjenu L, a ako ju ne radi, određena je kazna od jednog boda. U istom isječku, radnik s identifikacijskom oznakom G dao je zahtjev da treći dan ne radi smjenu E, a ako ju radi, određena je kazna od dva boda.

```
SECTION_SHIFT_ON_REQUESTS
# EmployeeID, Day, ShiftID, Weight
A,5,L,1

...

SECTION_SHIFT_OFF_REQUESTS
# EmployeeID, Day, ShiftID, Weight
G,3,E,2
```

Isječak 14: Isječak ulazne datoteke u kojoj su određeni zahtjevi radnika za i protiv smjene.

U programskom rješenju, kao što je vidljivo u isječku 15, ograničenje je modelirano s `ShiftOnRequest` i `ShiftOffRequest` razredima. Apstraktnom razredu `SoftConstraint` prosljeđuje se kazna kako bi se sve kazne mogle zbrojiti i različita rješenja usporediti prema kazni. U slučaju kršenja nekog od ograničenja, stvara se instanca razreda u kojoj je zabilježeno kojem radniku je koji dan prekršeno koje ograničenje.

```
data class ShiftOnRequest(val employeeId: String, val shiftRequest:
↪ ShiftRequest, val actual: String?) :
    SoftConstraint(shiftRequest.weight)

data class ShiftOnRequest(val employeeId: String, val shiftRequest:
↪ ShiftRequest, val actual: String?) :
    SoftConstraint(shiftRequest.weight)
```

Isječak 15: Isječak programskog modela koji modelira zahtjeve za smjenom i zahtjeve protiv smjene radnika.

5.2.2. Pokriće smjena

U formulaciji problema zadani su zahtjevi za egzaktan broj radnika za određeni dan i smjenu, tj. koliko radnika bi trebalo raditi određenu smjenu taj dan. Primjer navedenog ograničenja dan je u isječku 16. U tom isječku, nulti dan bi četiri radnika trebala raditi smjenu `E`. Ako nulti dan više od četiri radnika rade smjenu `A` tada je kazna za svakog radnika preko ograničenja jedan bod, a ako manje od četiri radnika rade smjenu `A` tada je kazna za svakog radnika ispod ograničenja sto bodova.

```
SECTION_COVER
# Day, ShiftID, Requirement, Weight for under, Weight for over
0,E,4,100,1
```

Isječak 16: Isječak ulazne datoteke u kojoj je određeno pokriće smjena.

U programskom rješenju, kao što je vidljivo u isječku 17, ograničenje je modelirano s `CoverConstraint` razredom. U slučaju kršenja ograničenja, stvara se instanca razreda u kojoj je zabilježeno koji zahtjev za pokrićem je prekršen, ima li više ili manje radnika od potrebnog te koji dan je prekršeno ograničenje.

```
data class CoverConstraint(
    val cover: Cover,
    val coverType: CoverType,
    val filled: Int,
    val day: Int
) : SoftConstraint(calculatePenalty(cover, coverType, filled))
```

Isječak 17: Isječak programskog modela koji modelira zahtjeve za smjenom i zahtjeve protiv smjene radnika.

5.3. GRASP - konstrukcijska faza

Algoritam započinje konstrukcijskom fazom algoritma GRASP u kojem se pronalazi rješenje koje zadovoljava sva tvrda ograničenja. Izrada rasporeda kreće od potpuno praznog rasporeda, rasporeda u kojem niti jednom radniku nije dodijeljena niti jedna smjena.

5.3.1. Podjela poslova

U konstrukcijskoj fazi algoritma vrši se podjela poslova tako da jedan posao odgovora pronalasku rasporeda jednog radnika. Broj radnika za koji se paralelno traži raspored moguće je unijeti kao parametar algoritma. Budući da ne postoje tvrda ograničenja između radnika moguće je njihove poslove izvršavati u potpunosti neovisno, ali u ovom slučaju se to nije pokazao kao najbolji pristup. Zbog toga, svi poslovi dijele memoriju čija uloga je objašnjena u idućim poglavljima.

Svaki posao modeliran je Javinim `Runnable` sučeljem i izvršava se preko `ExecutorService` razreda.

5.3.2. Problemi zadovoljavanja tvrdih ograničenja

Tijekom pretrage rješenja, u slučaju definicije susjedstva kao promjene jedne smjene jednom radniku, moguće je vršiti selekciju susjeda tako da se zadovolje sva tvrda ograničenja osim ograničenja minimalnog broja radnih minuta.

Početno rješenje ne zadovoljava ograničenje minimalnog broja radnih minuta radnika. Tijekom generiranja rasporeda nije moguće garantirati da će ograničenje minimalnog broja radnih minuta biti zadovoljeno. Raspored pojedinog radnika gradi se korak po korak te se u svakom koraku, rasporedu radnika mijenja jedna ili više smjena i nema garancije da algoritam neće otići u smjeru u kojem je raspored napravljen na takav način da nije moguće ostvariti ograničenje od minimalnog broja minuta.

Ograničenje minimalnog broja uzastopnih smjena moguće je zadovoljiti selekcijom susjeda na navedeni način, ali rješenje nije implementirano tako jer bi rezultiralo s konačnim rješenjem slabije kvalitete. Problem koji se javlja pri pokušaju zadovoljenja ograničenja minimalnog broja uzastopnih smjena je taj da ako bi se idući korak u generiranju rasporeda radnika gradio smjenu po smjenu tada bi, na početku generiranja, jedina dozvoljena mjesta za smjenu bila na početku i kraju perioda planiranja. Idući koraci bi tada bili isključivo na početku ili kraju perioda planiranja. Tada bi smjenama na početku i kraju perioda planiranja bilo dodijeljeno previše radnika, a smjena na sredini perioda planiranje premalo radnika. Ovaj problem javlja se zbog toga što u danim instancama radnici moraju raditi barem dva uzastopna dana.

Zbog gore navedenog, tijekom pretraživanja susjedstva prihvaćaju se rješenja koja moraju zadovoljavati sva tvrda ograničenja osim dva gore navedena.

5.3.3. Način pretrage

Pretraga za svakog radnika izvršava se zasebno, a susjedstvo koje se razmatra su jedna ili više promjena smjena jednog radnika.

Pretraga mogućih rješenja započinje tako da se za svaki dan planiranog perioda pronađu smjene koje radnik taj dan može raditi, a da zadovoljava sva ograničenja osim minimalnog broja radnih minuta. Jedini izuzetak se radi za rješenja koja uz ograničenje minimalnog broja radnih minuta ne zadovoljavaju i ograničenje minimalnog broja uzastopnih smjena. Takva rješenja spremaju se zbog uporabe u daljnjem koraku.

Nakon pretrage rješenja postoje dvije skupine rješenja:

– skup `A` - skup u kojem rješenja rasporeda opcionalno zadovoljavaju ograničenje minimalnog broja radnih minuta te zadovoljavaju sva ostala tvrda ograničenja

– skup `B` - skup u kojem rješenja rasporeda opcionalno zadovoljavaju ograničenje minimalnog broja radnih minuta i ne zadovoljavaju ograničenje minimalnog broja uzastopnih smjena te zadovoljavaju sva ostala tvrda ograničenja.

Sva rješenja koja su u skupu `A` kandidati su za idući raspored radnika. Tim rješenjima dodaju se i sva rješenja koja nastaju spajanjem rješenja iz skupa `A` i skupa `B` u blokove. Dodavanjem bloka smjena u početno rješenje mogu se prekršiti tvrda ograničenja te se takva rješenja ne uzimaju obzir kao kandidati za idući raspored radnika.

5.3.4. Spajanje rješenja u blokove

Prilikom pretrage susjedstva, ispitana su sva rješenja koja se od osnovnog rješenja razlikuju za najviše jednu smjenu. Za sva rješenja iz skupa `A` i iz skupa `B` zabilježeno je koja smjena je promijenjena u odnosu na osnovno rješenje.

Tada se generiraju blokovi duljine u ovisnosti o parametru `neighborhood.search_type`. Za svaku moguću duljinu bloka, na svim mogućim položajima u rasporedu radnika, radi se kartezijev produkt svih promijenjenih smjena tih dana. Za svaku permutaciju izrađuje se rješenje koje se evaluira te se prihvaća samo ako zadovoljava sva tvrda ograničenja ili jedino ograničenje koje ne zadovoljava je ograničenje minimalnog broja minuta.

U isječku 18, dan je pseudokod početka faze izgradnje rješenja. Varijabla `timesRestarted` bilježi broj ponovnog pokretanja pretraživanja. Nakon pronađenog rasporeda za pojedinog radnika, pronađeno rješenje se zapisuje u zajedničku mapu te se ažuriraju vrijednosti pokrića smjena što se događa u trinaestoj liniji isječka.

```
1  var latestSolution: SolutionBuilder
2
3  var timesRestarted = 0
4
5  do {
6      latestSolution = currentSolutionBuilder.copy()
7
8      latestSolution = findSolutionsForGivenEmployee(...,
        ↪  timesRestarted)
9
10     timesRestarted++
11 } while (latestSolution.breakingAnyHardConstraint())
12
13 updateCommonCoverMapAndCommonSolutionMap(...)
```

Isječak 18: Pseudokod faze izgradnje rješenja.

U isječku 19, dan je pseudokod metode `findSolutionsForGivenEmployee` koja pronalazi i odabire idući korak pretrage. Za svaki dan u periodu planiranja, pronalaze se sve smjene koje radnik može raditi taj dan što se događa u sedmoj liniji isječka.

Smjene su podijeljene u dvije grupe, one u kojoj rješenja zadovoljavaju sva tvrda ograničenja osim minimalnog broja uzastopnih smjena i one u kojoj sva rješenja zadovoljavaju sva tvrda ograničenja. Između devete i četrnaeste linije ispituje se jesu li pronađene smjene koje radnik može raditi na dan `day`, a ako su pronađene, sve smjene koje zadovoljavaju sva tvrda ograničenja dodaju se u kandidate za iduću smjenu, a sve pronađene smjene dodaju se u smjene od kojih se mogu stvarati blokovi smjena.

```
for (iteration in 0 until config.grasp.iterationsBeforeRestart){

    val solutions = LinkedList<SolutionBuilder>()

    for (day in employeeSchedule.indices) {

        val neighborSolutions = findShifts(...)

        if (neighborSolutions.isEmpty()) {
            dayToSolutionsThatCanBeMerged[day] = emptyList()
        } else {
            solutions.addAll(neighborSolutions[true])
            dayToSolutionsThatCanBeMerged[day] =
             ↪  neighborSolutions.values.flatten()
        }
    }
    solutions.addAll(mergeDaysToBlocks(...))
    currentSolution = solutions.chooseFromRcl(...)

    if (currentSolution.notBreakingAnyHardConstraint()) {
        break;
    }
}
```

Isječak 19: Pseudokod faze izgradnje rješenja - logika pamćenja i odabira rješenja, isječak metode `findSolutionsForGivenEmployee`.

5.3.5. Odabir rješenja

U sedamnaestoj liniji isječka 19, gradi se RCL te se odabire iduće rješenje. Način odabira rješenja opisan je u 20 isječku.

Ovisno o tome postoji li rješenje koje zadovoljava sva ograničenja, provodi se različiti postupak odabira susjeda. Ako postoji rješenje koje zadovoljava sva tvrda ograničenja, tada se u obzir uzimaju jedino rješenja koja zadovoljavaju sva tvrda ograničenja.

U slučaju da ne postoji rješenje koje zadovoljava sve tvrde uvjete, zadržavaju se samo rješenja koja ne smanjuju broj radnih minuta radnika budući da je to jedino ograničenje koje rješenje u ovoj fazi može ne zadovoljavati.

U slučaju da je izvršeno manje iteracija od parametra danog u konfiguracijskoj datoteci ili ako postoji rješenje koje zadovoljava sva tvrda ograničenja tada se vrši filtracija po poboljšanju rješenja po broju poteza. Jedan potez je jedna promijenjena

smjena u odnosu na osnovno rješenje, a ako je npr. dodan blok od tri smjene to se računa kao tri poteza.

```
val nonBreakingExists = this.any { it.hardConstraints.size == 0 }

var candidates = this.filter { it.hardConstraints.size == if
    (nonBreakingExists) 0 else 1 }

if (!nonBreakingExists) {
    candidates = filterByTimeImproving(candidates, baselineMinutes)
}


if (timesRestarted < config.grasp.iterationBeforeIgnoringPenalty ||
    nonBreakingExists) {
    candidates =
        filterByPenaltyImprovementPerAppliedMove(candidates,
        baselinePenalty, config)
}

return candidates.randomOrNull()
}
```

Isječak 20: Pseudokod odabira rješenja.

5.3.6. Primjer stanja tijekom pretraživanja

Algoritam započinje konstrukcijskom fazom algoritma GRASP u kojem se pronalazi rješenje koje zadovoljava sva tvrda ograničenja.

U konstrukcijskoj fazi algoritma vrši se podjela poslova tako da jedan posao odgovora jednom radniku. Budući da ne postoje tvrda ograničenja između radnika moguće je njihove poslove izvršavati u potpunosti neovisno. Broj radnika za koji se paralelno traži raspored moguće je unijeti kao parametar algoritma.

Međutim, meko ograničenje vezano uz pokriće smjena značajno utječe na kvalitetu konačnog rješenja te ga nije preporučljivo zanemariti jer će pojedinačno dobivena rješenja koja zadovoljavaju tvrde uvjete zajedno činiti loše rješenje.

Zbog toga, u implementaciji rješenja postoji dijeljena mapa u kojoj su agregatne informacije o smjenama radnika koji su do tog trenutka izračunati i podatci o pokriću smjena radnika.

Tijekom pronalaska rasporeda, primjer mogućeg rasporeda završenih radnika, informacija u zajedničkoj mapi te pokrića smjena dan je redom u tablicama 5.1, 5.3 i 5.2. Tijekom pretrage rješenja, za radnika `A` i radnika `B` je pronađeno rješenje te su po završetku svakog od njih u zajedničku mapu zabilježene smjene koje oni rade. Tada, prilikom izračuna kvalitete rješenja nekog nedovršenog radnika kod izračuna kazne za pokriće smjena u obzir se uzima njegov raspored smjena i raspored smjena dovršenih radnika.

Tablica 5.1: Primjer rasporeda završenih radnika.

radnik	dan 0	dan 1	dan 2
A		n	m
B	m		m

Tablica 5.2: Primjer ograničenja pokrića smjena.

smjena	dan 0	dan 1	dan 2
n	2	2	2
m	1	3	1

Tablica 5.3: Primjer informacija u zajedničkoj mapi na temelju završenih radnika iz tablice 5.1.

smjena	dan 0	dan 1	dan 2
n	0	1	0
m	1	0	2

Nakon što su pronađeni rasporedi koji zadovoljavaju tvrda ograničenja za svakog radnika, svi rasporedi su spojeni u jedan raspored koji sadrži sve rasporede te na kome se u idućem koraku izvršava lokalna pretraga.

5.4. GRASP - faza lokalne pretrage / VND

U fazi lokalne pretrage implementirano je nekoliko algoritama lokalne pretrage čiji je cilj smanjiti ukupnu kaznu rješenja na koju utječu tri ograničenja - ograničenje pokrića smjena, ograničenje zahtjeva za smjenom i ograničenje zahtjeva protiv smjene. U lokalnim pretragama ima najviše smisla fokusirati se na ograničenje pokrića smjene budući da u svim danim instancama kazna za premali broj radnika određenog dana u određenoj smjeni je barem red veličine veća od svih ostalih kazni za meka ograničenja.

5.4.1. Prvi algoritam lokalne pretrage

Prvi algoritam lokalne pretrage fokusira se na zadovoljavanje pokrića smjena. Algoritam radniku koji je slobodan ili radi neku smjenu na kojoj taj dan radi previše radnika dodjeljuje smjenu koja nije dodijeljena dovoljnom broju radnika te provjerava je li tim potezom prekršeno neko tvrdo ograničenje.

Prvo se pronalaze sva prekršena ograničenja egzaktnog broja smjena i to takva da je broj radnika manji od ograničenja. Pronađena ograničenja se silazno sortiraju po kazni. Za svako od prekršenih ograničenja, pronalaze se radnici koji taj dan nemaju dodijeljenu smjenu ili imaju dodijeljenu smjenu kojoj je taj dan dodijeljeno previše radnika. Svakom od radnika se pokušava dodijeliti smjena koja je ispod ograničenja egzaktnog broja smjena.

Ako su nakon dodjeljivanja te smjene nekom radniku sva tvrda ograničenja i dalje zadovoljena te je zadovoljeno ograničenje egzaktnog broja smjena koje je bilo prekršeno, nastavlja se postupak s idućim nezadovoljenim ograničenjem egzaktnog broja smjena. Ako nakon dodjele smjene nekom radniku i dalje postoji premali broj radnika koji radi tu smjenu, pretraga radnika koji mogu raditi tu smjenu se nastavlja.

Modificirani isječak algoritma dan je u isječku 21. U prvom retku pronalaze se prekršena ograničenja pokrića gdje je broj radnika manji od željenog te se sortiraju prema kazni. U petom retku, pronalaze se radnici koji su slobodni na dan kada postoji kršenje ograničenja pokrića te se sortiraju uzevši u obzir zahtjeve za i protiv smjena.

```
val brokenUnderCovers = filterUnderCoverAndSortByPenalty()

for (brokenUnderCover in brokenUnderCovers) {

    val nurses = findNurses(...)

    for (nurse in nurses) {

        val tempSolution = bestSolution.copy()

        tempSolution.setShift(
                nurse,
                brokenUnderCover.day,
                brokenUnderCover.cover.shiftId
        )

        if (tempSolution.isNotBreakingHardConstraint()) {
            bestSolution = tempSolution

            if (bestSolution.totalPenalty >
             ↪  tempSolution.totalPenalty) {
                bestSolution = tempSolution
                }

            if (isUnderCoverFixed(tempSolution, brokenUnderCover)) {
                break
            }
        }
    }
}
```

Isječak 21: Modificirani isječak programskog koda koji opisuje 1. algoritam lokalne pretrage.

Primjer izvođenja algoritma prikazan je tablicama 5.4, 5.5, 5.6, 5.7. U tablici 5.4 označeno je da prvog dana postoji manjak radnika za smjenu `D`. Radi se o jednom radniku manjka što nosi kaznu od 100 bodova. U tablici 5.6 prikazan je raspored radnika prije provođenja pretrage, a u tablici 5.7 raspored nakon provođenja pretrage.

Tijekom izvođenja algoritma pronađeno je da je `radnik 0` slobodan prvog dana te mu je postavljena smjena `D` kako bi se zadovoljilo ograničenje što je vidljivo u tablici 5.5.

Tablica 5.4: Kazne za kršenje ograničenja pokrića prije primjene lokalne pretrage.

Smjena \ Dan	0	1	2
E	0	0	0
D	0	100	0
N	0	0	0

Tablica 5.5: Kazne za kršenje ograničenja pokrića nakon primjene lokalne pretrage.

Smjena \ Dan	0	1	2
E	0	0	0
D	0	0	0
N	0	0	0

Tablica 5.6: Primjer rasporeda prije primjene lokalne pretrage.

Radnik \ Dan	0	1	2
0			N
1	N	E	D
2	D		N
3	E	N	E

Tablica 5.7: Primjer rasporeda nakon primjene lokalne pretrage.

Radnik \ Dan	0	1	2
0		D	N
1	N	E	D
2	D		N
3	E	N	E

5.4.2. Drugi algoritam lokalne pretrage

Drugi algoritam lokalne pretrage fokusira se na zadovoljavanje pokrića smjena. Algoritam radniku kojemu je dodijeljena smjena na dan kada je previše radnika dodijeljena ista smjena, uklanja tu smjenu toga dana, a dodjeljuje smjenu kojoj nije dodijeljen dovoljan broj radnika, tog ili nekog drugog slobodnog dana tog radnika.

Modificirani isječak algoritma dan je u isječku 22. U prvom retku pronalaze se prekršena ograničenja pokrića, sortiraju se prema kazni i grupiraju prema tipu kršenja ograničenja (ima li radnika više ili manje od željenog). Tada se za svako prekršeno ograničenje pokrića gdje je broj radnika bio manji od željenog prolazi kroz sva prekršena ograničenja pokrića gdje je broj radnika bio veći od željenog. Tada se, u retku 6, pronalaze radnici koji na dan kada ima previše radnika za određenu smjenu rade tu smjenu te oni koji na dan kada ima premalo radnika za određenu smjenu ne rade tu smjenu. Dodatno, radnici se sortiraju uzevši u obzir njihove zamjene za smjenu i protiv smjene tako da radnici koji bi više smanjili kaznu vezanu uz te zahtjeve se ispituju ranije.

Za pronađene radnike, redom, mijenja se raspored te se ispituje je li promjenama došlo dok kršenja nekog tvrdog ograničenja. Ako nije, a rješenje ima manju kaznu

od trenutnog najbolje, sprema se. Ako je to rješenje riješilo manjak radnika za tom smjenu tog dana, zaustavlja se pretraga za tom smjenom toga dana.

```
val brokenCovers: Map<CoverType, List<CoverConstraint>> =
    sortBrokenCoversByPenaltyAndGroupByCoverType()

for (brokenUnderCover in brokenCovers[CoverType.UNDER]) {
    ocloop@ for (brokenOverCover in brokenCovers[CoverType.OVER]) {

        val nurses = findNurses(...)

        for (nurse in nursesThatWorkOnOverCoveredShift) {
            val tempSolutionBuilder = initialSolution.copy()
            tempSolutionBuilder.setShifts(
                listOf(
                    Triple(
                        nurse,
                        brokenOverCover.day,
                        null
                    ), Triple(
                        nurse,
                        brokenUnderCover.day,
                        brokenUnderCover.cover.shiftId
                    )
                )
            )
            if (tempSolution.isNotBreakingAnyHardConstraints()) {
                if (tempSolution.totalPenalty <
                    bestSolution.totalPenalty) {
                    bestSolution = tempSolution
                }

                if (isUnderCoverFixed(tempSolution,
                    brokenUnderCover)) {
                    break@ocloop
                }
            }
        }
    }
}
```

Isječak 22: Modificirani isječak programskog koda koji opisuje 2. algoritam lokalne pretrage.

Primjer izvođenja algoritma prikazan je tablicama 5.8, 5.9, 5.10, 5.11. U tablici 5.8 vidljivo je da nedostaje jedan radnik koji radi smjenu `D` prvog dana te da postoji jedan radnik viška koji radi smjenu `N` drugog dana. Pretragom radnika kojima je dodijeljena smjena koja je taj dan dodijeljena previše radnika, za radnika `radnik 0` pronađena je smjena `N` drugog dana. Umjesto te smjene, taj dan mu je dodijeljen slobodan dan, a prvog dana mu je dodijeljena smjena `D` kako bi se zadovoljilo ograničenje pokrića smjena `D` drugog dana.

Tablica 5.8: Kazne za kršenje ograničenja pokrića prije primjene lokalne pretrage.

Smjena \ Dan	0	1	2
E	0	0	0
D	0	100	0
N	0	0	1

Tablica 5.9: Kazne za kršenje ograničenja pokrića nakon primjene lokalne pretrage.

Smjena \ Dan	0	1	2
E	0	0	0
D	0	0	0
N	0	0	0

Tablica 5.10: Primjer rasporeda prije primjene lokalne pretrage.

Radnik \ Dan	0	1	2
0	N		N
1	N	E	D
2	D		N
3	E	N	E

Tablica 5.11: Primjer rasporeda nakon primjene lokalne pretrage.

Radnik \ Dan	0	1	2
0	N	D	
1	N	E	D
2	D		N
3	E	N	E

5.4.3. Vertikalna zamjena

Vertikalna zamjena fokusira se na smanjenje kazne vezanih uz zahtjeve za i protiv smjene. Algoritam za danu veličinu bloka (2, 3, 4, 5) zamjeni smjene dvaju radnika u određenom rasponu čiji početak je nasumično odabran. Uz veličinu bloka, uveden je dodatan parametar koji ograničava pretragu isključivo na vikende tako da se zamijene smjene jednog vikenda s drugim.

```kotlin
val blockRange = generateBlockToSwap(instance)

val employeeId1 = entries.random().key

var employeeId2: String
do {
    employeeId2 = entries.random().key
} while (employeeId1 == employeeId2)

initialSolution.swapBlocksBetweenTwoEmployees(blockRange,
    employeeId1, employeeId2)
```

Isječak 23: Modificirani isječak programskog koda koji opisuje vertikalnu zamjenu smjena.

5.4.4. Horizontalna zamjena

Horizontalna zamjena fokusira se na smanjenje kazni pokrića smjene. Algoritam za danu veličinu bloka (2, 3, 4, 5) zamjeni smjene unutar istog radnika u određenom rasponu čiji početak je nasumično odabran. Uz veličinu bloka, uveden je dodatan parametar koji ograničava pretragu isključivo na vikende tako da se zamijene smjene jednog vikenda s drugim.

```kotlin
val rangesToSwap = generateRangesToSwap(instance)

val employeeToShift = entries.random().key

initialSolution.swapBlocksInSameEmployee(blocksToSwap,
    employeeToShift)
```

Isječak 24: Modificirani isječak programskog koda koji opisuje horizontalnu zamjenu smjena.

5.5. Konačni algoritam

U isječku 25 dan je pseudokod konačnog algoritma lokalne pretrage. U metodi `loopVerticalLs1Ls2` ponavlja se izvođenje nasumično odabrane vertikalne zamjene te prvog i drugog algoritma lokalne pretrage sve dok se kazna smanjuje. U metodi `loopHorizontal` se nasumično odabire vrsta horizontalne zamjene te se primjenjuje na trenutno rješenje. Konačni algoritam se prekida kada se određeni broj puta zaredom, koji je dan u konfiguracijskoj datoteci, ne smanji kazna.

```
1  do {
2      val iterationStartPenalty = bestSolution.totalPenalty
3
4      currentSolution = loopVerticalLs1Ls2(currentSolution, instance)
5      bestSolution = setNewBestIfImproved(currentSolution,
       ↪  bestSolution)
6
7      repeat(config.horizontalSearch.repeat) {
8          currentSolution = loopHorizontal(currentSolution, instance)
9      }
10     bestSolution = setNewBestIfImproved(currentSolution,
       ↪  bestSolution)
11
12     timesNotImproved = if (bestSolution.totalPenalty <
       ↪  iterationStartPenalty) 0 else ++timesNotImproved
13
14 } while (timesNotImproved <
   ↪  config.variableNeighborDescent.breakAfterTimesNotImproved
15 )
```

Isječak 25: Modificirani isječak programskog koda koji opisuje konačni algoritam lokalne pretrage.

6. Konfiguracija programskog rješenja

6.1. Primjer konfiguracije programskog rješenja

Programsko rješenje konfigurirano je pomoću biblioteke Hoplite [7]. Put do konfiguracijske datoteke potrebno je predati kao argument programskom rješenju.

Primjer konfiguracijske datoteke dan je u isječcima 26 i 27.

```
1  instance: 1
2  instances_path: src/main/resources/instances/
3  search_algorithms: GRASP, VND
4
5  grasp:
6      solutions_by_employee_path:
       ↪  src/main/resources/solutions-by-employee/
7      solutions_write_path: src/main/resources/solutions-grasp-write/
8      iterations_before_restart: 10x
9      iteration_before_ignoring_penalty: 10
10     penalty_alpha: 0.25
11     threads: 6
12
13 neighborhood:
14     searchType: mid
15     includeNoShift: true
16
17 local_search:
18     solutions_read_path: src/main/resources/solutions-read/
19     solutions_write_path:
       ↪  src/main/resources/solutions-grasp-ls-write/
```

Isječak 26: Primjer konfiguracije programskog rješenja.

```
horizontal_search:
    repeat: 10

variable_neighbor_descent:
    break_after_times_not_improved: 25
```

Isječak 27: Primjer konfiguracije programskog rješenja (nastavak).

6.2. Opis konfiguracije programskog rješenja

U idućim tablicama, navedeni su svi parametri konfiguracijske datoteke te njihova objašnjenja.

U tablici 6.1 navedeni su općeniti parametri algoritma. Parametru `search_algorithms` može se predati samo parametar `VNS` te će tada izvršiti samo lokalna pretraga, a početno rješenje će se pročitati iz parametra koji se nalazi u tablici 6.6. Ako se parametru `search_algorithms` predaju parametri `GRASP`, `VNS` onda će se izvršiti pronalazak početnog rješenja i lokalna pretraga nad nađenim rješenjem.

Tablica 6.1: Opis općenitih konfiguracijskih parametara programskog rješenja.

Parametar	Objašnjenje parametra
instance	broj instance
instances_path	put do instanci
solutions_path	put do rješenja
search_algorithms	odabir algoritma pretraživanja

U tablici 6.2 navedeni su parametri GRASP algoritma. Na put `solutions_by_employee_path` zapisuju koraci izrade rasporeda te elementi koji su promijenjeni između dva rješenja za lakše praćenje i razumijevanje postupka izgradnje.

Tablica 6.2: Opis konfiguracijskih parametara algoritma GRASP.

Parametar	Objašnjenje parametra
solutions_by_employee_path	put do rješenja (detaljni postupak izgradnje rješenja)
solutions_write_path	put do rješenja
iterations_before_restart	broj iteracija prije ponovnog pokretanja algoritma u fazi izgradnje u odnosu na broj dana planiranog perioda
iteration_before_ignoring_penalty	broj iteracija prije zanemarivanja kazne rješenja u fazi izgradnje
penalty_alpha	parametar α za određivanje rješenja koja ulaze u RCL
threads	broj dretvi koje se paralelno izvršavaju u konstrukcijskoj fazi izgradnje rješenja

U tablici 6.3 navedeni su parametri pretraživanja susjedstva. Moguće vrijednosti parametra `search_type` su `min`, `mid` i `max` koji određuju maksimalnu veličinu bloka u fazi izgradnje rješenja. Parametar `min` ograničava veličinu bloka na minimalni broj uzastopnih dana koje radnik mora raditi, parametar `mid` ograničava veličinu bloka na srednju vrijednost između minimalnog i maksimalnog broja uzastopnih (zaokruženo na niže), a parametar `max` ograničava veličinu bloka na maksimalan broj uzastopnih dana koje radnik može raditi. Parametar `include_no_shift` određuje hoće li se u generiranim blokovima smjena nalaziti slobodni dani ili će svim danima u bloku biti dodijeljene smjene.

Tablica 6.3: Opis konfiguracijskih parametara istraživanja susjedstva.

Parametar	Objašnjenje parametra
search_type	određuje veličinu bloka rješenja
include_no_shift	uključuju li blokovi slobodan dan

U tablici 6.4 navedeni su parametri pretraživanja susjedstva. S puta `solutions_read_path` se učitava rješenja ako je parametar `search_algorithms` iz tablice 6.1 postavljen na `VND`.

Tablica 6.4: Opis konfiguracijskih parametara lokalne pretrage.

Parametar	Objašnjenje parametra
solutions_read_path	put do učitanog rješenja
solutions_write_path	put do generiranog rješenja

U tablici 6.5 naveden je parametar horizontalne lokalne pretrage.

Tablica 6.5: Opis konfiguracijskih parametara horizontalne pretrage.

Parametar	Objašnjenje parametra
repeat	broj uzastopnih pokretanja horizontalne pretrage

U tablici 6.6 naveden je parametar algoritma VND.

Tablica 6.6: Opis konfiguracijskih parametara algoritma VND.

Parametar	Objašnjenje parametra
break_after_times_not_improved	broj uzastopnih iteracija bez poboljšanja rješenja potrebnih za zaustavljanje algoritma

7. Usporedba i moguće nadogradnje rješenja

U isječku 28 dani su relevantni konfiguracijski podatci. Do devetnaeste instance za parametar `neighborhood.searchType` korištena je vrijednost `max`, a za više instance korištena je vrijednost `mid` zbog zahtjeva za radnom memorijom. Pretraga je vršena na laptopu s Intel i7-8750H procesorom te 16 gigabajta radne memorije, a maksimalna veličina gomile postavljena je na 10 gigabajta.

```
grasp:
    iterations_before_restart: 10x
    iteration_before_ignoring_penalty: 10
    penalty_alpha: 0.25
    threads: 6

neighborhood:
    searchType: max
    includeNoShift: true

horizontal_search:
    repeat: 25

variable_neighbor_descent:
    break_after_times_not_improved: 10
```

Isječak 28: Korišteni konfiguracijski podatci algoritma.

U tablici 7.1 navedene su instance problema, broj tjedana planiranja, broj radnika, broj tipova smjena, najbolje rješenje koje je pronašao opisani algoritam te najbolje rješenje koje je pronađeno.

Za svaku instancu rješenje se pretraživalo jedan sat. Ako je pretraživanje završilo prije isteka od jednog sata, pokrenuto je ponovno te je uzeto u obzir ako je završilo u

roku od jednog sata od prvog pokretanja. Ako rješenje nije pronađeno u označeno je s N/A,a zvjezdicom su označene instance za koje su pronađena rješenja, ali je pretraga trajala više od jednog sata.

Tablica 7.1: Opis instanci problema i usporedba pronađenog s najboljim rješenjem.

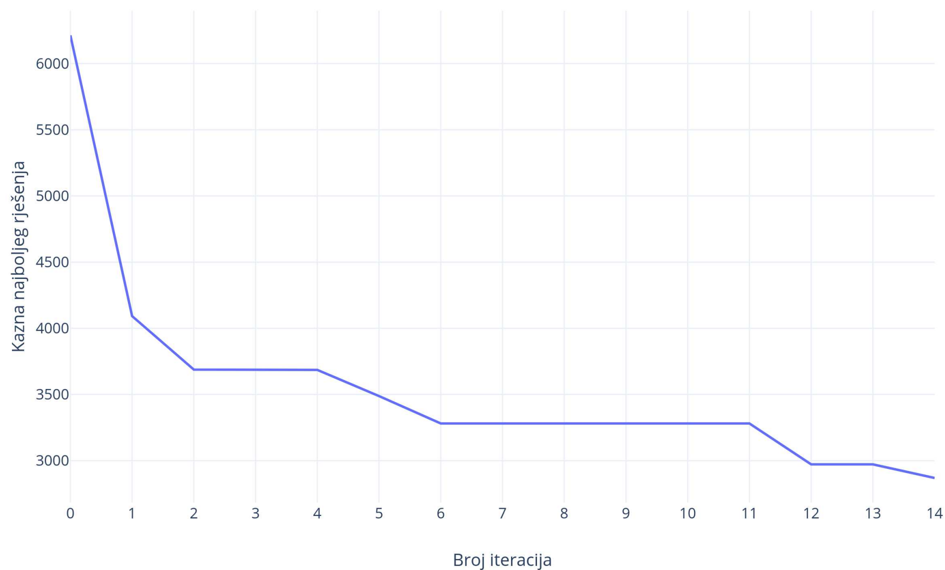
Instanca	Tjedana	Radnika	Tipova smjena	Pronađeno rješenje	Najbolje rješenje
1	2	8	1	609	607
2	2	14	2	944	828
3	2	20	3	1229	1001
4	4	10	2	1854	1716
5	4	16	2	1484	1143
6	4	18	3	2720	1950
7	4	20	3	1770	1056
8	4	30	4	3166	1300
9	4	36	4	1242	439
10	4	40	5	5864	4631
11	4	50	6	4726	3443
12	4	60	10	8232	4040
13	4	120	18	8118	1348
14	6	32	4	2858	1278
15	6	45	6	11105	3831
16	8	20	3	5523	3225
17	8	32	4	10222	5746
18	12	22	3	9083	4459
19	12	40	5	11016*	3149
20	26	50	6	N/A	4769
21	26	100	8	N/A	21133
22	52	50	10	N/A	30244
23	52	100	16	N/A	17428
24	52	150	32	N/A	42463

Na kvalitetu rješenja algoritma najveći utjecaj ima broj tipova smjena. Veliki broj tipova smjena značajno povećava prostor pretraživanja te ima veće memorijske zahtjeve što usporava pronalazak rješenja koje zadovoljava sva tvrda ograničenja ili onemogućava pronalazak rješenja zbog nedovoljno radne memorije. Taj problem je po-

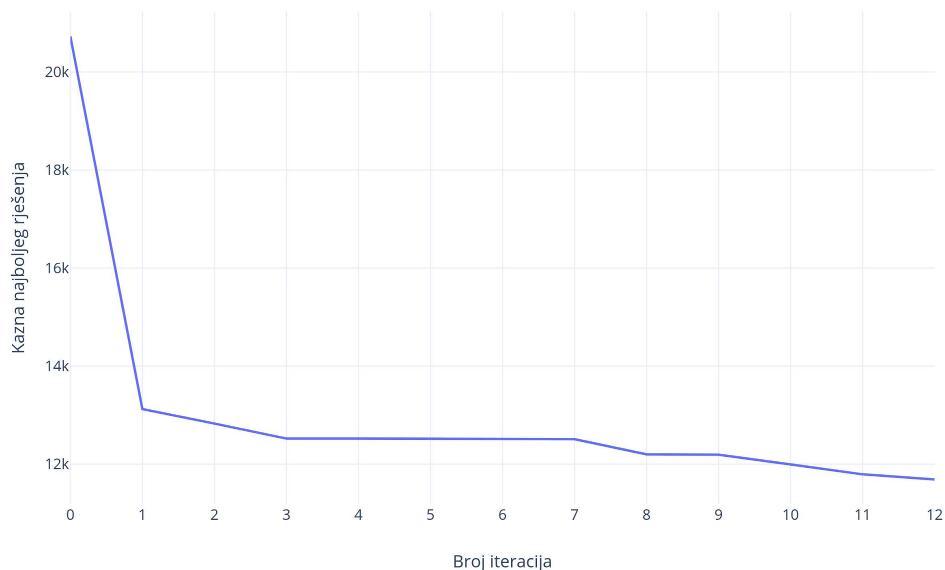
sebno izražen u dugim periodima planiranja.

U fazi izgradnje rješenja, problem mogu biti radnici koji imaju visoki zahtjev za minimalnim brojem minuta budući da nema garancije da će se on ostvariti u postupku pretrage rješenja te se u nekim slučajevima ponavlja pretraga za nekog radnika što usporava pronalazak rješenja. Broj radnika ne igra značajnu ulogu u pronalasku rješenja budući da se rješenja za svakog radnika nalaze zasebno.

Na slikama 7.1 i 7.2 dani su primjeri kretanja kazne najboljeg rješenja kroz iteracije u fazi lokalne pretrage. Određeni broj iteracija na kraju izvođenja algoritma s najmanjom kaznom je uklonjen iz prikaza zbog preglednosti grafikona.



Slika 7.1: Kretanje kazne kroz iteracije algoritma u fazi lokalne pretrage za 5. instancu problema.



Slika 7.2: Kretanje kazne kroz iteracije algoritma u fazi lokalne pretrage za 15. instancu problema.

Nadogradnju rješenja moguće je postići ubrzanjem postojeće implementacije. Postojeća implementacija u slučaju bilo kakve promjene rješenja ponovno ispituje sva ograničenja, a u nekim slučajevima bi se neke od provjera mogle preskočiti i time ubrzati izvođenje algoritma. Prilikom odabira kandidata za idući korak mogao bi se pamtiti manji broj informacija čime bi se smanjila potreba za radnom memorijom. Dodatno, može se izvršiti pretvorba svih naziva u brojeve kako bi se u implementaciji rješenja mape mogle pretvoriti u liste čime bi se ubrzalo izvođenje rješenja.

Najveći utjecaj na konačnu kvalitetu rješenja imaju lokalne pretrage i njihov poredak izvođenja. U implementaciju se mogu dodati dodatne lokalne pretrage ili promijeniti broj i/ili red izvođenja postojećih lokalnih pretraga.

8. Zaključak

Cilj ovog rada je izraditi kvalitetan raspored radnika u prihvatljivom vremenskom periodu. Za instance problema uzete su standardno korišteni primjeri.

U fazi izgradnje rješenja algoritma GRASP, implementiran je algoritam koji paralelno gradi rješenja radnika. Algoritam se pokazao dobrim te pronalazi rješenje koje zadovoljava sva tvrda ograničenja u roku od nekoliko minuta, osim za najveće instance problema. Trenutna implementacija algoritma koristi više radne memorije nego je potrebno što postaje problem kod velikih instanci problema.

U fazi lokalne pretrage algoritma GRASP, iskorišteno je više algoritama koji imaju različitu ulogu u pronalaženju optimalnog rješenja. Taj dio algoritma nije memorijski zahtjevan, ali ga nije moguće jednostavno paralelizirati. Ta faza pretrage traje značajno dulje od faze izgradnje rješenja.

U budućnosti moguće je optimizirati trenutnu implementaciju algoritma koja u fazi izgradnje rješenja koristi previše radne memorije. U fazi lokalne pretrage, moguće je dodati dodatne algoritme lokalne pretrage ili promijeniti poredak i broj izvođenja trenutno implementiranih algoritama.

LITERATURA

[1] Nurse Rostering Benchmark Instances. `http://www.schedulingbenchmarks.org/nrp/`.

[2] Mohammed Abdelghany, Zakaria Yahia, i Amr B Eltawil. A new two-stage variable neighborhood search algorithm for the nurse rostering problem. *RAIRO Oper. Res.*, 55(2):673–687, Ožujak 2021.

[3] Tim Curtois i Rong Qu. Computational results on new staff scheduling benchmark instances. *tech. report*, 2014.

[4] Lucas Kletzander i Nysret Musliu. Solving the general employee scheduling problem. *Computers & Operations Research*, 113:104794, 2020. doi: 10.1016/j.cor.2019.104794.

[5] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[6] Yi Qu i Timothy Curtois. Solving the multi-activity shift scheduling problem using variable neighbourhood search. *Proceedings of the 9th International Conference on Operations Research and Enterprise Systems*, 2020. doi: 10.5220/0009118102270232.

[7] Sam Sam. Hoplite, 2022. URL `https://github.com/sksamuel/hoplite`.

[8] Nina Skorin-Kapov i Lea Skorin-Kapov. Greedy algorithms; Metaheuristics: basic concepts; Local search. GRASP; Improving local search. `https://www.fer.unizg.hr/predmet/hmo_a/materijali`, 2021.

Rješavanje problema raspoređivanja medicinskog osoblja korištenjem metaheuristika

Sažetak

Rad proučava problem raspoređivanja medicinskog osoblja u bolnicama, odnosno općeniti problema izrade rasporeda radnika. U radu je dana teorijska podloga korištenih algoritama i detaljni opis vrste problema koji se rješava. Predstavljen je algoritam koji koristi konstrukcijsku fazu GRASP (Greedy randomized adaptive search procedure) algoritma za pronalazak rješenja koje zadovoljava sva tvrda ograničenje te koji koristi fazu lokalne pretrage u kojoj je primijenjen algoritam VND (Variable Neighbourhood Descent) za pronalazak boljeg rješenja. Postupak algoritma GRASP prilagođen je tipu problema i instancama problema. Performanse algoritma uspoređene su na često korištenim instancama s najbolje pronađenim rješenjima.

Ključne riječi: GRASP, VND, problem izrade rasporeda, metaheuristike

Solving the nurse rostering problem using metaheuristics

Abstract

The paper studies nurse scheduling problem, i.e. the general problem of scheduling workers. The paper provides a theoretical basis for the algorithms used and a detailed description of the type of problem to be solved. An algorithm uses the construction phase of the GRASP (Greedy randomized adaptive search procedure) algorithm to find a solution that satisfies all hard constraints and uses a local search phase in which the VND (Variable Neighborhood Descent) algorithm is applied to find a better solution. The GRASP algorithm procedure is adapted to the type of the problem and the instances of the problem. The performance of the algorithm was compared on frequently used instances with the best found solutions.

Keywords: GRASP, VND, nurse scheduling problem, metaheuristics

Popis slika

4.1. Ilustracija postupka pretraživanja algoritmom VND, izvor ilustracije: [8]. 6

7.1. Kretanje kazne kroz iteracije algoritma u fazi lokalne pretrage za 5. instancu problema. 34
7.2. Kretanje kazne kroz iteracije algoritma u fazi lokalne pretrage za 15. instancu problema. 35

POPIS TABLICA

5.1. Primjer rasporeda završenih radnika. 20
5.2. Primjer ograničenja pokrića smjena. 20
5.3. Primjer informacija u zajedničkoj mapi na temelju završenih radnika iz tablice 5.1. 20
5.4. Kazne za kršenje ograničenja pokrića prije primjene lokalne pretrage. 23
5.5. Kazne za kršenje ograničenja pokrića nakon primjene lokalne pretrage. 23
5.6. Primjer rasporeda prije primjene lokalne pretrage. 23
5.7. Primjer rasporeda nakon primjene lokalne pretrage. 23
5.8. Kazne za kršenje ograničenja pokrića prije primjene lokalne pretrage. 25
5.9. Kazne za kršenje ograničenja pokrića nakon primjene lokalne pretrage. 25
5.10. Primjer rasporeda prije primjene lokalne pretrage. 25
5.11. Primjer rasporeda nakon primjene lokalne pretrage. 25

6.1. Opis općenitih konfiguracijskih parametara programskog rješenja. . . 29
6.2. Opis konfiguracijskih parametara algoritma GRASP. 30
6.3. Opis konfiguracijskih parametara istraživanja susjedstva. 30
6.4. Opis konfiguracijskih parametara lokalne pretrage. 31
6.5. Opis konfiguracijskih parametara horizontalne pretrage. 31
6.6. Opis konfiguracijskih parametara algoritma VND. 31

7.1. Opis instanci problema i usporedba pronađenog s najboljim rješenjem. 33

POPIS PROGRAMSKIH ISJEČAKA

1. Pseudokod algoritma GRASP. 3
2. Pseudokod lokalne pretrage rješenja. 5
3. Pseudokod VND algoritma. 7
4. Isječak ulazne datoteke u kojoj je navedeno ograničenje da ako radnik `i-ti` dan radi smjenu `L`, `i+1` dan ne može raditi smjenu `E`. 9
5. Isječak programskog modela koji modelira ograničenje da neke smjene ne smiju slijediti druge smjene. 9
6. Isječak ulazne datoteke u kojoj su navedena ograničenja za radnika s identifikacijskom oznakom `A`. 9
7. Isječak programskog modela koji modelira maksimalan broj puta koji radnik može raditi određenu smjenu. 10
8. Isječak programskog modela koji modelira ograničenje minimalnog i maksimalnog broja minuta radnika. 10
9. Isječak programskog modela koji modelira ograničenje minimalnog i maksimalnog broja uzastopnih smjena. 11
10. Isječak programskog modela koji modelira ograničenje minimalnog broja uzastopnih slobodnih dana. 11
11. Isječak programskog modela koji modelira ograničenje maksimalnog broja radnih vikenda. 12
12. Isječak ulazne datoteke u kojoj su određeni slobodni dani radnika. . . 12
13. Isječak programskog modela koji modelira ograničenje slobodnih dana za radnika. 12
14. Isječak ulazne datoteke u kojoj su određeni zahtjevi radnika za i protiv smjene. 13
15. Isječak programskog modela koji modelira zahtjeve za smjenom i zahtjeve protiv smjene radnika. 13
16. Isječak ulazne datoteke u kojoj je određeno pokriće smjena. 14

17. Isječak programskog modela koji modelira zahtjeve za smjenom i zahtjeve protiv smjene radnika. 14
18. Pseudokod faze izgradnje rješenja. 17
19. Pseudokod faze izgradnje rješenja - logika pamćenja i odabira rješenja, isječak metode `findSolutionsForGivenEmployee`. 18
20. Pseudokod odabira rješenja. 19
21. Modificirani isječak programskog koda koji opisuje 1. algoritam lokalne pretrage. 22
22. Modificirani isječak programskog koda koji opisuje 2. algoritam lokalne pretrage. 24
23. Modificirani isječak programskog koda koji opisuje vertikalnu zamjenu smjena. 26
24. Modificirani isječak programskog koda koji opisuje horizontalnu zamjenu smjena. 26
25. Modificirani isječak programskog koda koji opisuje konačni algoritam lokalne pretrage. 27
26. Primjer konfiguracije programskog rješenja. 28
27. Primjer konfiguracije programskog rješenja (nastavak). 29
28. Korišteni konfiguracijski podatci algoritma. 32

Dodatak A
Format ulazne datoteke

U nastavku je dan primjer ulazne datoteke programskog rješenja, koji je modificirana (skraćena) verzija 2. instance problema koja je javno dostupna [1].

```
1  # This is a comment. Comments start with #
2  SECTION_HORIZON
3  # All instances start on a Monday
4  # The horizon length in days:
5  7
6
7  SECTION_SHIFTS
8  # ShiftID, Length in mins, Shifts which cannot follow this shift |
   ↪  separated
9  E,480,
10 L,480,E
11
12 SECTION_STAFF
13 # ID, MaxShifts, MaxTotalMinutes, MinTotalMinutes,
   ↪  MaxConsecutiveShifts, MinConsecutiveShifts,
   ↪  MinConsecutiveDaysOff, MaxWeekends
14 A,E=7|L=7,2160,1680,5,2,2,1
15 B,E=7|L=7,2160,1680,5,2,2,1
16 C,E=7|L=7,2160,1680,5,2,2,1
17 D,E=7|L=0,2160,1680,5,2,2,1
18 E,E=0|L=7,2160,1680,5,2,2,1
19 F,E=7|L=7,2160,1680,5,2,2,1
20 G,E=7|L=7,2160,1680,5,2,2,1
21 H,E=7|L=7,2160,1680,5,2,2,1
22 I,E=7|L=7,2160,1680,5,2,2,1
23 J,E=7|L=7,2160,1680,5,2,2,1
24 K,E=0|L=7,1080,600,5,1,1,1
25 L,E=0|L=7,1080,600,5,1,1,1
```

```
26 M,E=7|L=7,1080,600,5,1,1,1
27 N,E=7|L=7,1080,600,5,1,1,1
28
29
30 SECTION_DAYS_OFF
31 # EmployeeID, DayIndexes (start at zero)
32 A,3
33 B,1
34 C,2
35 E,1
36 H,3
37 I,0
38 K,5
39 L,2
40 N,6
41
42 SECTION_SHIFT_ON_REQUESTS
43 # EmployeeID, Day, ShiftID, Weight
44 A,5,L,1
45 A,6,L,1
46 D,1,E,1
47 D,2,E,1
48 D,3,E,1
49 E,3,L,1
50 E,4,L,1
51 E,5,L,1
52 E,6,L,1
53 F,3,L,3
54 F,4,L,3
55 F,5,L,3
56 I,2,L,3
57 I,3,L,3
58 L,3,L,1
59 L,4,L,1
60 M,3,L,1
61 M,4,L,1
62 M,5,L,1
63 M,6,L,1
64 N,0,E,2
65 N,1,E,2
66 N,2,E,2
67
68 SECTION_SHIFT_OFF_REQUESTS
```

```
69 # EmployeeID, Day, ShiftID, Weight
70 G,3,E,2
71 G,4,E,2
72 G,5,E,2
73 G,6,E,2
74 H,1,L,2
75 J,1,E,1
76 J,2,E,1
77 J,3,E,1
78 J,4,E,1
79 J,5,E,1
80
81 SECTION_COVER
82 # Day, ShiftID, Requirement, Weight for under, Weight for over
83 0,E,4,100,1
84 0,L,4,100,1
85 1,E,4,100,1
86 1,L,3,100,1
87 2,E,3,100,1
88 2,L,6,100,1
89 3,E,5,100,1
90 3,L,4,100,1
91 4,E,3,100,1
92 4,L,4,100,1
93 5,E,5,100,1
94 5,L,5,100,1
95 6,E,5,100,1
96 6,L,5,100,1
```