INTELIGENTNI SISTEMI

as. ms Vladimir Jocović
as. ms Adrian Milaković

METODI ZADOVOLJENJA OGRANIČENJA

02

CSP

„Every problem has a solution.
You just have to be creative enough to find it.“
- Travis Kalanick

METODI ZADOVOLJENJA OGRANIČENJA (CSP)

U standardnim algoritmima pretraživanja:

- Stanje je „crna kutija“. Postoji startno stanje, ciljno stanje i ostala stanja u koja se u procesu pretraživanja može dospeti.
- Operatori/akcije omogućavaju prelazak iz jednog u drugo stanje.
- Test cilja proverava da li je trenutno stanje jedno od ciljnih stanja.

U metodima zadovoljenja ograničenja:

- Stanje se definiše promenljivama Xi koje mogu uzimati vrednosti iz domena Di.
- Test cilja proverava da li su dodeljene **vrednosti** svakoj od **promenljivih** u skladu sa **ograničenjima** nametnutih od strane samog problema, tj. da li su nametnuta ograničenja zadovoljena (**konzistentna dodela**).

HILLCLIMBING pretraga

Planinarenje (eng. **Hillclimbing**) je vrsta lokalne pretrage, koja se može primeniti u rešavanju CSP problema. Na početku se svim promenljivama na određeni način (slučajnim ili heurističkim izborom) dodele početne vrednosti iz njihovih domena i takvo stanje se proglasi kao startno.

Startno stanje predstavlja **potpunu** dodelu koja ignoriše ograničenja nametnuta problemom (ne predstavlja konzistentnu dodelu).

Zatim se na osnovu startnog (trenutnog) stanja generiše novo susedno stanje izborom **promenljive** koja ima **nerazrešena ograničenja** sa drugim promenljivama i izborom vrednosti za tu promenljivu tako da novo stanje bude bolje od trenutnog (da krši manje ograničenja). Može se težiti tome da novo stanje bude najbolje od svih mogućih susednih stanja trenutnog stanja, ali ne mora nužno, već je dovoljno da je bolje od trenutnog stanja. Novo stanje se proglašava za trenutno.

Korak izbora promenljive sa nerazrešenim ograničenjima i njene vrednosti je potrebno ponavljati dok god je moguć prelazak u novo, bolje stanje.

HILLCLIMBING ALGORITAM

```
# state - stanje problema
# vars - promenljive u problemu
# domains - vrednosti koju svaka promenljiva može da dobije
# constraints - ograničenja koja važe

def hill_climbing(state, vars, domains, constraints):
  while True:
    if is_goal(state):
      break
    next_state = next_better_neighbour(state, vars, domains,
                                                    constraints)
    if next_state is None or not is_better(next_state, state):
      break
    state = next_state
return state
```

HILLCLIMBING pretraga

Da li planinarenje garantuje pronalaženje optimalnog rešenja?

- Ne, jer možda sva stanja nisu dostižna iz inicijalno izabranog startnog stanja. Algoritam bi mogao da se „zaglavi“ u lokalno optimalnom stanju koje predstavlja rešenje ili da uopšte ne pronađe rešenje. Takođe, algoritam bi mogao da se zaglavi na platou (ravnici), gde nema boljih ili lošijih susednih stanja.

Kako izbeći problem lokalno optimalnog stanja?

- Ponoviti pretragu iz novog slučajno izabranog startnog stanja. Na taj način mogu da se istraže različiti delovi prostora pretraživanja.

Da li dozvoliti prelaz u stanja koja su lošija od stanja iz kojeg vršimo prelaz?

- Metoda Simuliranog kaljenja (eng. Simulated Annealing) dozvoljava prelazak u stanja koja su lošija od trenutnog u cilju bežanja iz lokalno optimalnih rešenja.

SIMULATED ANNEALING ideja

Simulirano kaljenje (eng. Simulate annealing) je vrsta lokalne pretrage, koja je dobila naziv po procesu zagrevanja i kontrolisanog hlađenja materijala u metalurgiji radi njihove lakše obrade.

Startno stanje predstavlja potpunu dodelu koja ignoriše ograničenja nametnuta problemom (ne predstavlja konzistentnu dodelu).

Zatim se na slučajan način izabere novo susedno stanje u koje je moguće preći iz trenutnog stanja. Izvrši se evaluacija kvaliteta susednog stanja u odnosu na trenutno stanje i, ukoliko je novo stanje bolje, dozvoljava se prelazak u njega, a, ukoliko je novo stanje lošije, u zavisnosti od parametara sistema (temperatura, broj koraka) dozvoljava se prelaz u takvo stanje (npr. ako je temperatura visoka).

Prethodni korak se ponavlja dok god se ne dosegne ciljno stanje ili dok ne dođe do promene vrednosti parametara sistema koji zahtevaju prestanak njegovog rada (dostignut maksimalni broj koraka ili dostignuta minimalna temperatura – „ohlađen materijal“).

SIMULATED ANNEALING ALGORITAM

```
# state - stanje problema
# vars - promenljive u problemu
# domains - vrednosti koju svaka promenljiva može da dobije
# constraints - ograničenja koja važe
# max_steps - maksimalan broj koraka algoritma
# temperature - trenutna temperatura sistema
# params - ostali parametri sistema

def simulated_annealing(state, vars, domains, constraints):
  for step in range(max_steps):
    if is_goal(state):
      break
    next_state = next_neighbour(state, vars, domains,constraints)
    if is_better(next_state, state) or \
       is_allowed(next_state, state, temperature, step, params):
      state = next_state
    temperature = decrease_temperature(temperature, step, params)
  return state
```

BACKTRACKING pretraga

Pretraga unazad (eng. **Backtracking**) je obična **DFS** pretraga, čija je **dubina limitirana** (jer je dodela vrednosti komutativna) na broj promenljivih Xi u problemu koji se rešava.

Stanje u problemu se definiše dodelama vrednosti (koje su do tada napravljene) promenljivama. **Inicijalno stanje** je prazna dodela.

Funkcija prelaza iz stanja u stanje vrši dodelu vrednosti jednoj od promenljivih koja nema dodeljenu vrednost. Takva dodela ne sme da krši nijedno ograničenje nametnuto samim problemom. Ukoliko takva dodela ne postoji, onda je neophodno vratiti se korak unazad (*backtrack*) i promeniti dodeljenu vrednost prethodnoj promenljivoj kojoj je vrednost dodeljena ili vratiti se na prethodnu promenljivu, ukoliko su isprobane sve vrednosti.

Ciljni test proverava da li je trenutno stanje ciljno, tj. da li je dodela vrednosti promenljivama kompletna i konzistentna.

BACTRACKING ALGORITAM

```
def backtrack_search(vars, domains, solution, lvl, constraints):
  if lvl == len(vars):
    return True  # all variables are assigned

  v = vars[lvl]
  for val in domains[v]:
    if is_consistent_assignment(v, val, vars, domains, constraints):
      solution[v] = val  # assign value
      new_dom = copy.deepcopy(domains)  # copy domain
      new_dom[v] = [val]  # modify new domain
      if backtrack_search(vars, new_dom, solution, lvl+1, constraints):
        return True  # solution is found
      solution[v] = None  # backtrack
  return False
```

BACKTRACKING pretraga - unapređenje

Kako izabrati promenljivu čija će vrednosti biti promenjena?

- Izabrati najograničeniju promenljivu, tj. promenljivu kojoj je ostao najmanji broj legalnih vrednosti u domenu (**Minimum Remaining Values**). Ideja je završiti „pogrešnu pretragu“ što pre.
- Ukoliko postoji više takvih promenljivih, onda izabrati onu koja učestvuje u najvećem broju ograničenja sa ostalim promenljivama (**Most Constraining/Constrained Variable**). Ukoliko ponovo postoji više takvih promenljivih, izabrati promenljivu na slučajan način.

Kako izabrati novu vrednost izabrane promenljive?

- Izabrati vrednost koja najmanje ograničava (**Least Constraining Value**) ostale promenljive sa kojom je izabrana promenljiva u ograničenju, tj. onu vrednost koja će izbaciti najmanji broj vrednosti iz domena ostalih promenljivih.

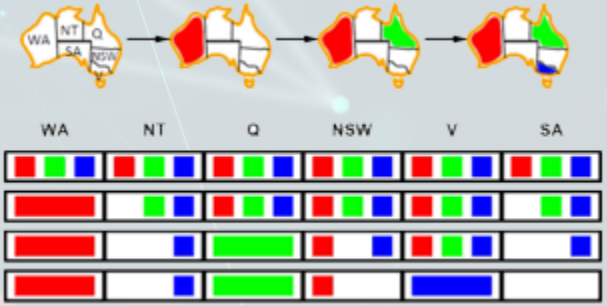
BACTRACKING ALGORITAM

```
def backtrack_search(vars, domains, solution, lvl, constraints):
  if lvl == len(vars):
    return True

  v = get_most_constrained_variable(vars, domains, constraints)
  sort_domain_least_constraining(v, vars, domains, constraints)
  for val in domains[v]:
    if is_consistent_assignment(v, val, vars, domains, constraints):
      solution[v] = val
      new_dom = copy.deepcopy(domains)
      new_dom[v] = [val]
      if backtrack_search(vars, new_dom, solution, lvl+1, constraints):
        return True
      solution[v] = None
  return False
```

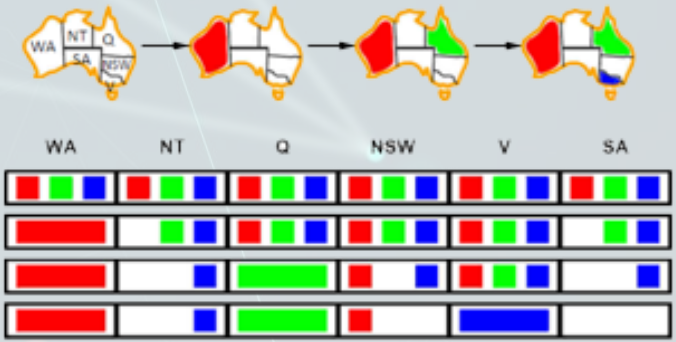
FORWARD CHECKING

Ideja **Forward checking**-a je da se (nakon što se dodeli vrednost izabranoj promenljivoj) ažuriraju domeni svih onih promenljivih koje su u ograničenju sa izabranom promenljivom, a koje još nisu dobile vrednost. U slučaju da u nekom trenutku domen barem jedne promenljive **postane prazan**, odnosno ne postoji nijedna legalna vrednost koja se može dodeliti toj promenljivoj, onda se pretraga na tom nivou prekida i neophodno je vratiti se korak unazad (*backtrack*). Na taj način se ranije otkriva „pogrešna putanja“.



FORWARD CHECKING

Forward checking ažurira domene promenljivih sa kojima je izabrana promenljiva u ograničenju i koje još uvek nisu dobile vrednosti, ali ne omogućava ranu detekciju za sve „pogrešne putanje“. Npr. teritorije NT i SA ne mogu biti plave boje i to je moglo biti otkriveno već prilikom dodele zelene boje teritoriji Q. Prilikom korišćenja *Forward checking*-a „pogrešna putanja“ je otkrivena kasnije, prilikom dodele plave boje teritoriji V.



FC-BACTRACKING ALGORITAM

```
def backtrack_search(vars, domains, solution, lvl, constraints):
  if lvl == len(vars):
    return True

  v = get_most_constrained_variable(vars, domains, constraints)
  sort_domain_least_constraining(v, vars, domains, constraints)
  for val in domains[v]:
    if is_consistent_assignment(v, val, vars, domains, constraints):
      solution[v] = val
      new_dom = copy.deepcopy(domains)
      new_dom[v] = [val]
      for var in vars:
        if var != v and are_constrained(v, var, constraints):
          update_domain(new_dom[var], v, val, constraints)
      if backtrack_search(vars, new_dom, solution, lvl+1, constraints):
        return True
      solution[v] = None
  return False
```

CONSTRAINT PROPAGATION

Glavna ideja propagacije ograničenja je da se prilikom izbora vrednosti izabrane promenljive uradi provera validnosti domena za sve promenljive koje su u ograničenju sa izabranom promenljivom, a koje još uvek nisu dobile vrednost. Ukoliko se ažurira domen takvih promenljivih, onda je istu proveru neophodno uraditi i za promenljive koje su u ograničenjima sa takvim promenljivama, a za koje važi da još uvek nisu dobile vrednost itd. Ova provera domena se izvršava dok god ima promenljivih čije domene treba proveriti na taj način. Ovim metodom je ranije moguće otkriti „pogrešnu putanju“.

CSP problem se može predstaviti grafom, gde su čvorovi promenljive, a grane ograničenja koja povezuju promenljive koje su u ograničenju. U slučaju ograničenja u kojima učestvuju više od dve promenljive, uvodi se poseban tip čvorova koji povezuju promenljive (čvorove) u ograničenju.

CONSTRAINT PROPAGATION

Konzistencija čvora – sve vrednosti u domenu promenljive/čvora zadovoljavaju sva unarna ograničenja te promenljive. Nije potrebna nikakva propagacija ovih informacija kroz CSP graf, već samo restrikcija domena promenljive u skladu sa datim unarnim ograničenjima te promenljive.

Konzistencija luka – najjednostavnija forma propagacije ograničenja koja čini dve promenljive povezane granom (lukom) ograničenja konzistentnim. Luk koji povezuje dve promenljive X i Y i to u smeru X -> Y je konzistentan ako važi da za svaku vrednost x iz domena DX promenljive X važi da postoji barem jedna vrednost y iz domena DY promenljive Y takva da ne krši dato binarno ograničenje. Ukoliko to nije slučaj, vrednost x se mora obrisati iz domena DX kako bi luk ograničenja X -> Y bio konzistentan. Brisanje vrednosti x iz domena DX promenljive X se mora propagirati kroz CSP graf svim promenljivama Z iz skupa setZ, koje su u ograničenju Z -> X sa promenljivom X, a kojima još uvek nije dodeljena vrednost.

AC3 ALGORITAM

```
def arc_consistency(vars, domains, constraints):
  arc_list = get_all_arcs(vars, domains, constraints)
  while arc_list:
    x, y = arc_list.pop(0)
    x_vals_to_del = []
    for val_x in domains[x]:
      y_no_val = True
      for val_y in domains[y]:
        if satisfies_constraint(val_x, val_y, x, y, constraints):
          y_no_val = False
          break
      if y_no_val:
        x_vals_to_del.append(val_x)
    if x_vals_to_del:
      domains[x] = [v for v in domains[x] if v not in x_vals_to_del]
      if not domains[x]:
        return False
      for v in vars:
        if v != x and are_constrained(v, x, constraints):
          arc_list.append((v, x))
  return True
```

ARC-BACTRACKING ALGORITAM

```
def backtrack_search(vars, domains, solution, lvl, constraints):
  if lvl == len(vars):
    return True

  v = get_most_constrained_variable(vars, domains, constraints)
  sort_domain_least_constraining(v, vars, domains, constraints)
  for val in domains[v]:
    if is_consistent_assignment(v, val, vars, domains, constraints):
      solution[v] = val
      new_dom = copy.deepcopy(domains)
      new_dom[v] = [val]
      if not arc_consistency(vars, new_dom, constraints):
        solutions[v] = None
        continue
      if backtrack_search(vars, new_dom, solution, lvl+1, constraints):
        return True
      solution[v] = None
  return False
```

CONSTRAINT PROPAGATION

Propagacija ograničenja u nekim situacijama može rešiti problem bez korišćenja klasične pretrage prostora problema, ali ne možemo uvek računati na to.

Najčešće se propagacija ograničenja koristi kao preprocesirajući korak, pre samog procesa pretrage, sa ciljem da značajno redukuje domene promenljivih na osnovu datih ograničenja problema. Ovakvo korišćenje propagacije ograničenja je jeftino i potencijalno olakšava pretragu koja sledi.

Propagacija ograničenja se može koristiti i dinamički, u toku same pretrage. Taj izbor se može pokazati skupljim u slučaju da postoji veliki broj povrataka unazad (*backtrack*), jer je u tom slučaju potrebno restaurirati redukovane domene.

Zadatak 1 – Problem rasporeda časova

★★☆☆☆

Boško, Dražen, Vladimir i Adrian drže časove istog dana u isto vreme. Svaki nastavnik ima svoj skup omiljenih sala, što zbog njihove opremljenosti, što zbog lokacije. Dva nastavnika ne mogu da drže čas u istoj učionici. Želje nastavnika su date u nastavku:

Boško – 61, 309

Dražen – 57, 59, 61, Lola-AMF

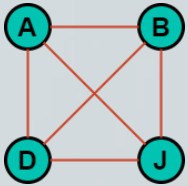
Vladimir – 61, 309

Adrian – 61, Lola-AMF

Nacrtati graf susedstva, uvesti sve potrebne promenljive i ograničenja i primenom metoda zadovoljenja ograničenja dodeliti svakom nastavniku učionicu u kojoj će da drži čas, ako se koristi pretraga unapred (*forward checking*). Napisati redosled izbora promenljivih, ako pri izboru biramo promenljivu sa minimalnim brojem preostalih vrednosti (leksikografski u slučaju više takvih promenljviih), a pri izboru vrednosti promenljivih biramo vrednost po redosledu navođenja.

Zadatak 1 – Rešenje

Graf susedstva dat je na slici desno.



Čvorovima koji su u ograničenju (susednim čvorovima) ne može biti dodeljena ista vrednost.

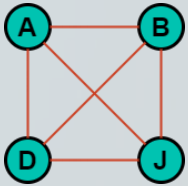
Boško – { 61, 309 }
Dražen – { 57, 59, 61, Lola-AMF }
Vladimir – { 61, 309 }
Adrian – { 61, Lola-AMF }

Biramo promenljivu kojoj ćemo dodeliti vrednost. Po heuristici iz zadatka promenljive Boško, Vladimir i Adrian imaju najmanji broj preostalih vrednosti u domenu te se leksikografski bira promenljiva Adrian.

Vrednost koju dodeljujemo promenljivoj Adrian biramo po redosledu navođenja, tako da će joj biti dodeljena vrednost 61.

Zadatak 1 – Rešenje

Trenutno stanje nakon dodele vrednosti (korak 1):



Boško – { 61, 309 }
Dražen – { 57, 59, 61, Lola-AMF }
Vladimir – { 61, 309 }
Adrian – 61

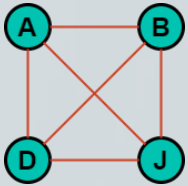
Forward checking izbacuje vrednost 61 iz svih suseda čvora Adrian.

Boško – { ~~61~~, 309 }
Dražen – { 57, 59, ~~61~~, Lola-AMF }
Vladimir – { ~~61,~~ 309 }
Adrian – 61

Promenljive Boško i Vladimir imaju najmanji broj preostalih vrednosti u domenu te leksikografski biramo promenljivu Boško i dodeljujemo joj jedinu preostalu vrednost 309.

Zadatak 1 – Rešenje

Trenutno stanje nakon dodele vrednosti (korak 2):



Boško – 309
Dražen – { 57, 59, 61, Lola-AMF }
Vladimir – { 309 }
Adrian – 61

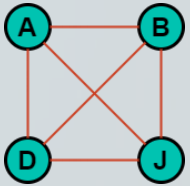
Forward checking izbacuje vrednost 309 iz svih suseda čvora Boško.

Boško – 309
Dražen – { 57, 59, Lola-AMF }
Vladimir – { ~~309~~ }
Adrian – 61

Kako je domen vrednosti dostupan promenljivoj Vladimir prazan, radimo *backtracking*. Za promenljivu Boško ne postoji druga vrednost koju možemo izabrati te idemo jedan nivo nazad. Za promenljivu Adrian pored sale 61 može se izabrati i Lola-AMF pa se ta vrednost dodeljuje promenljivoj Adrian.

Zadatak 1 – Rešenje

Trenutno stanje nakon *backtracking-a* i dodele vrednosti (korak 3):



Boško – { 61, 309 }
Dražen – { 57, 59, 61, Lola-AMF }
Vladimir – { 61, 309 }
Adrian – Lola-AMF

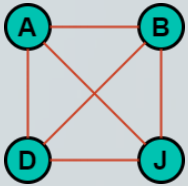
Forward checking izbacuje vrednost Lola-AMF iz svih suseda čvora Adrian.

Boško – { 61, 309 }
Dražen – { 57, 59, 61, ~~Lola-AMF~~ }
Vladimir – { 61, 309 }
Adrian – Lola-AMF

Promenljive Boško i Vladimir imaju najmanji broj preostalih vrednosti u domenu te leksikografski biramo promenljivu Boško i dodeljujemo joj prvu salu po redosledu želja, salu 61.

Zadatak 1 – Rešenje

Trenutno stanje nakon dodele vrednosti (korak 4):



Boško – 61
Dražen – { 57, 59, 61 }
Vladimir – { 61, 309 }
Adrian – Lola-AMF

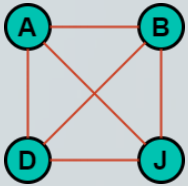
Forward checking izbacuje vrednost 61 iz svih suseda čvora Boško.

Boško – 61
Dražen – { 57, 59, ~~61~~ }
Vladimir – { ~~61~~, 309 }
Adrian – Lola-AMF

Promenljiva Vladimir ima najmanji broj preostalih vrednosti u domenu te joj dodeljujemo jedinu preostalu vrednost 309.

Zadatak 1 – Rešenje

Trenutno stanje nakon dodele vrednosti (korak 5):



Boško – 61
Dražen – { 57, 59 }
Vladimir – 309
Adrian – Lola-AMF

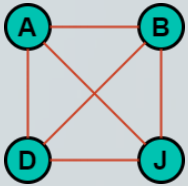
Forward checking ne izbacuje nijednu novu vrednost i ne menja stanje.

Boško – 61
Dražen – { 57, 59 }
Vladimir – 309
Adrian – Lola-AMF

Promenljiva Dražen ima najmanji broj preostalih vrednosti u domenu te joj dodeljujemo prvu dostupnu vrednost 57.

Zadatak 1 – Rešenje

Trenutno stanje nakon dodele vrednosti (korak 6):



Boško – 61
Dražen – 57
Vladimir – 309
Adrian – Lola-AMF

Forward checking ne izbacuje nijednu novu vrednost i ne menja stanje.

Boško – 61
Dražen – 57
Vladimir – 309
Adrian – Lola-AMF

Svim promenljivama su dodeljene vrednosti i problem je time rešen!
Za samostalnu vežbu: Uraditi zadatak bez *forward checking* poboljšanja.
Za samostalnu vežbu: Uraditi zadatak sa *arc-consistency* poboljšanjem.
Uporediti broj koraka u kojima je problem rešen.

Zadatak za samostalnu vežbu - Raspored

★★☆☆☆

Boško, Dražen, Vladimir i Adrian drže časove studentima četvrte godine. Svaki nastavnik ima svoj skup željenih termina u kojima bi preferirao da drži nastavu. Nastavnici ne smeju da imaju preklapanja u časovima jer studenti ne mogu da budu na dva mesta u isto vreme. Želje nastavnika su:

Boško – 11:00 – 13:00, 12:00 – 14:00

Dražen – 11:00 – 13:00, 13:00 – 15:00, 14:00 – 16:00

Vladimir – 10:00 – 12:00, 14:00 – 16:00, 18:00 – 20:00

Adrian – 10:00 – 12:00, 11:00 – 13:00, 12:00 – 14:00, 13:00 – 15:00

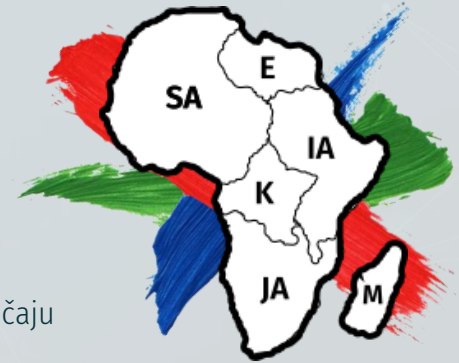
Nacrtati graf susedstva, uvesti sve potrebne promenljive i ograničenja i primenom metoda zadovoljenja ograničenja dodeliti svakom nastavniku učionicu u kojoj će da drži čas, ako se koristi pretraga unapred (*forward checking*). Napisati redosled izbora promenljivih ako pri izboru biramo promenljivu sa minimalnim brojem preostalih vrednosti (leksikografski u slučaju više takvih promenljviih), a pri izboru vrednosti promenljivih biramo vrednosti po redosledu navođenja u željama.

Zadatak 2 – Problem bojenja mape

★★★☆☆

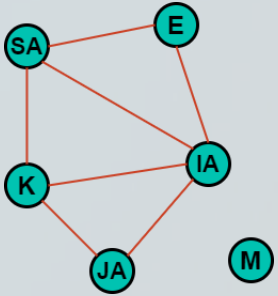
Data je mapa Afrike sa 6 regiona: Severna Afrika, Egipat, Istočna Afrika, Kongo, Južna Afrika i Madagaskar. Na raspolaganju su tri boje: crvena, zelena i plava. Susedni regioni ne mogu biti obojeni istom bojom. Nacrtati graf susedstva, uvesti sve potrebne promenljive i ograničenja i primenom metoda zadovoljenja ograničenja dodeliti svakom regionu jednu od mogućih boja, ako se koristi pretraga unapred (*forward checking*) uz primenu konzistentnosti lukova (*arc consistency*) nakon dodele vrednosti.

Napisati redosled izbora promenljivih, ako pri izboru biramo promenljivu sa minimalnim brojem preostalih vrednosti (ili promenljivu koja učestvuje u najviše ograničenja sa ostalim promenljivama kojima vrednost nije dodeljena ili leksikografski u slučaju više takvih promenljivih), a pri izboru vrednosti promenljivih biramo vrednost koja najmanje ograničava (crvena, zelena, plava u slučaju više takvih).



Zadatak 2 – Rešenje

Graf susedstva prikazan je na slici desno.



Ograničenje je da susedni regioni ne mogu biti obojeni istom bojom.

SA – { R, G, B }
E – { R, G, B }
K – { R, G, B }
IA – { R, G, B }
JA – { R, G, B }
M – { R, G, B }

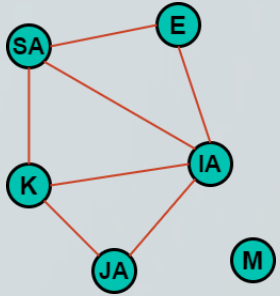
Biramo promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Sve promenljive imaju jedak broj preostalih vrednosti pa biramo onu koja učestvuje u najviše ograničenja sa promenljivama koje nemaju dodeljenu vrednost, a to je IA (4 suseda).

Vrednost koju dodeljujemo biramo tako da najmanje ograničava druge susede. Kako sve moguće vrednosti utiču na vrednosti sva četiri suseda, biramo crvenu boju.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti i FC (korak 1):

SA – { ~~R~~, G, B }
E – { ~~R~~, G, B }
K – { ~~R~~, G, B }
IA – R
JA – { ~~R~~, G, B }
M – { R, G, B }



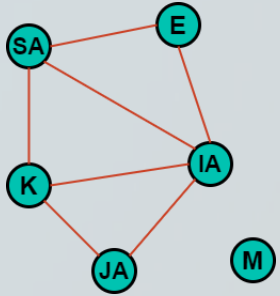
Proveravamo konzistentnosti lukova SA-E, SA-K, E-SA, K-SA, K-JA, JA-K.
Svi lukovi su isti po skupu vrednosti sa obe strane pa ćemo na jednom primeru objasniti:

SA-E:
Ako za SE uzmemo vrednost G, promenljivoj E ostaje barem jedna vrednost (B).
Analogno, važi i obrnuto, ako za SE uzmemo vrednost B, promenljivoj E ostaje barem jedna vrednost (G).
Dakle, luk je konzistentan i ne izbacujemo nijednu vrednost iz domena SA.

Zadatak 2 – Rešenje

Trenutno stanje nakon AC (korak 1):

SA – { G, B }
E – { G, B }
K – { G, B }
IA – R
JA – { G, B }
M – { R, G, B }



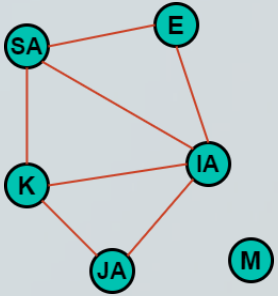
Biramo novu promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Sve promenljive imaju jedak broj preostalih vrednosti pa biramo onu koja učesvuje u najviše ograničenja sa drugim promenljivama, a to su SA (2 suseda kojima nisu dodeljene vrednosti) i K (2 suseda). Leksikografski biramo K.

Vrednost koju dodeljujemo biramo tako da najmanje ograničava druge susede. Kako sve moguće vrednosti utiču na vrednosti oba suseda, biramo zelenu boju.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti i FC (korak 2):

SA – { ~~G~~, B }
E – { G, B }
K – G
IA – R
JA – { ~~G~~, B }
M – { R, G, B }



Proveravamo konzistentnosti lukova SA-E i E-SA.

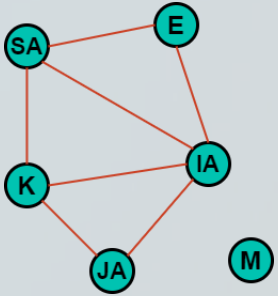
SA-E:
Ako za SE uzmemo vrednost B, promenljivoj E ostaje barem jedna vrednost (G).
Dakle, luk je konzistentan i ne izbacujemo nijednu vrednost iz domena SA.

E-SA:
Ako za E uzmemo vrednost G, promenljivoj SA ostaje barem jedna vrednost (B).
Ako za E uzmemo vrednost B, promenljivoj SA ne ostaje nijedna vrednost.
Dakle, luk nije konzistentan i izbacujemo B iz domena promenljive E.

Zadatak 2 – Rešenje

Trenutno stanje nakon prve iteracije AC (korak 2):

SA – { B }
E – { G, ~~B~~ }
K – G
IA – R
JA – { B }
M – { R, G, B }



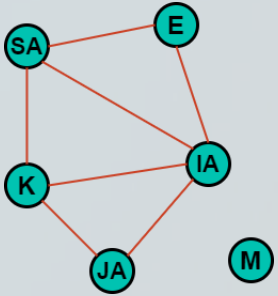
Proveravamo konzistentnost svih lukova ka čvoru E jer je njegov domen izmenjen.

SA-E:
Ako za SE uzmemo vrednost B, promenljivoj E ostaje barem jedna vrednost (G).
Dakle, luk je konzistentan i ne izbacujemo nijednu vrednost iz domena SA.

Zadatak 2 – Rešenje

Trenutno stanje nakon druge iteracije AC (korak 2):

SA – { B }
E – { G }
K – G
IA – R
JA – { B }
M – { R, G, B }



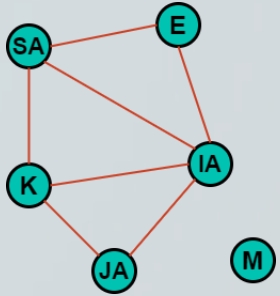
Biramo novu promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Promenljive SA, E i JA imaju jedak broj preostalih vrednosti pa biramo onu koja učestvuje u najviše ograničenja sa drugim promenljivama, a to su SA (1 sused) i E (1 sused). Leksikografski biramo E.

Vrednost koju dodeljujemo biramo kao jedinu preostalu opciju G.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti, FC i AC (korak 3):

SA – { B }
E – G
K – G
IA – R
JA – { B }
M – { R, G, B }



Forward checking nije izbacio nijednu vrednost iz domena suseda.

Arc consistency nije izbacio nijednu vrednost ni iz čijeg domena.

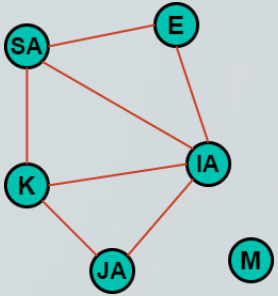
Biramo novu promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Promenljive SA i JA imaju jedak broj preostalih vrednosti, a kako nijedna nema susede kojima nisu dodeljene vrednosti, leksikografski biramo JA.

Vrednost koju dodeljujemo biramo kao jedinu preostalu opciju B.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti, FC i AC (korak 4):

SA – { B }
E – G
K – G
IA – R
JA – B
M – { R, G, B }



Forward checking nije izbacio nijednu vrednost iz domena suseda.

Arc consistency nije izbacio nijednu vrednost ni iz čijeg domena.

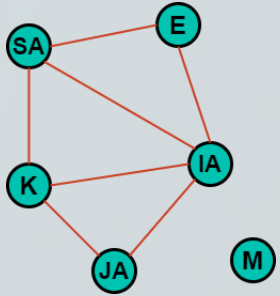
Biramo novu promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Promenljiva SA ima najmanji broj preostalih vrednosti pa nju biramo.

Vrednost koju dodeljujemo biramo kao jedinu preostalu opciju B.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti, FC i AC (korak 5):

SA – B
E – G
K – G
IA – R
JA – B
M – { R, G, B }



Forward checking nije izbacio nijednu vrednost iz domena suseda.

Arc consistency nije izbacio nijednu vrednost ni iz čijeg domena.

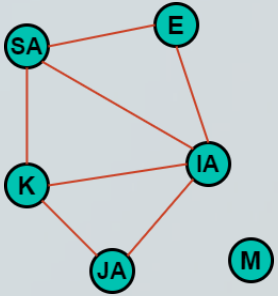
Biramo novu promenljivu kojoj ćemo dodeliti vrednost po heuristici iz postavke. Jedino je preostala promenljiva M pa nju biramo.

Vrednost koju dodeljujemo biramo po redosledu navođenja pa biramo crvenu boju.

Zadatak 2 – Rešenje

Trenutno stanje nakon dodele vrednosti, FC i AC (korak 6):

SA – B
E – G
K – G
IA – R
JA – B
M – R



Forward checking nije izbacio nijednu vrednost iz domena suseda.

Arc consistency nije izbacio nijednu vrednost ni iz čijeg domena.

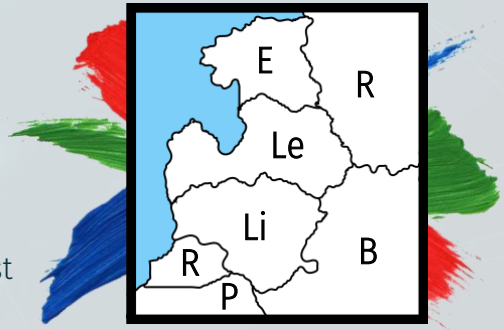
Sve promenljive imaju dodeljene vrednosti pa je problem rešen!

Zadatak za samostalnu vežbu - Baltik

★★★☆☆

Data je mapa Baltičkog regiona sa 6 zemalja: Rusija (2 regiona), Beloruskija, Estonija, Letonija, Litvanija i Poljska. Na raspolaganju su četiri boje: crvena, zelena, plava i žuta. Susedne zemlje ne mogu biti obojene istom bojom. Rusija je odabrala crveni boju, Estonija ne želi da bude žute boje, a Belorusija ne želi da bude plave boje. Unarana ograničenja mogu se odmah primeniti. Nacrtati graf susedstva, uvesti sve potrebne promenljive i ograničenja i primenom metoda zadovoljenja ograničenja dodeliti svakom regionu jednu od mogućih boja, ako se koristi pretraga unapred (*forward checking*) uz primenu konzistentnosti lukova (*arc consistency*) nakon dodele vrednosti.

Napisati redosled izbora promenljivih, ako pri izboru biramo promenljivu sa minimalnim brojem preostalih vrednosti (ili promenljivu koja učestvuje u najviše ograničenja sa ostalim promenljivama kojima vrednost nije dodeljena ili leksikografski u slučaju više takvih promenljivih), a pri izboru vrednosti promenljivih biramo vrednost koja najmanje ograničava (crvena, zelena, plava u slučaju više takvih).



Zadatak za samostalnu vežbu – Bioskop

★★★☆☆

Miki Maus, Mini Maus, Paja Patak, Pata, Šilja i Pluton su pošli u bioskop. U bioskopskoj sali se u svakom redu nalazi šest mesta numerisanih sleva na desno brojevima od 1 do 6. Družina je dobila jedan od redova u sali i svako od njih je izrazio svoje želje za sedenjem. Svako može sedeti samo na jednom mestu i na jednom mestu može sedeti samo jedan. Želje družine su date u nastavku:

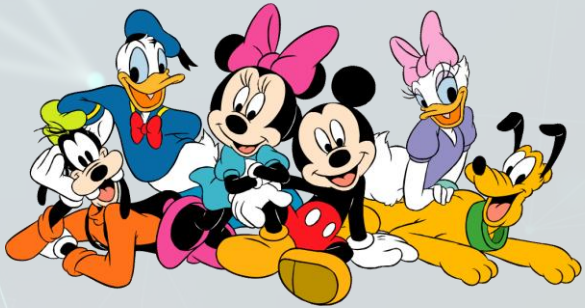
- Miki i Mini moraju da sede jedno pored drugog.
- Paja i Pata moraju da sede jedno pored drugog.
- Između Šilje i Mini mora da postoji minimalno dva mesta razmaka jer Šilja voli da komentariše film sa onima u blizini, a Mini ne voli da komentariše film i smeta joj priča u toku filma te oboma ne odgovara da sede jedno blizu drugog.
- Između Pate i Mikija mora da postoji minimalno dva mesta razmaka jer Pata ponekad plače u toku filma, a Mikiju to smeta te im ne odgovara da sede jedno blizu drugog.
- Pata ne može da sedi na krajevima reda (sedišta 1 i 6) jer joj smetaju prolaznici koji ulaze i izlaze u toku filma.

Zadatak za samostalnu vežbu – Bioskop

★★★☆☆

Pri izboru promenljive bira se ona sa minimalnim brojem preostalih vrednosti, a u slučaju više takvih promenljivih birati po sledećem redosledu: Pata, Paja, Šilja, Mini, Pluton, Miki. Pri izboru vrednosti promenljive bira se ona sa najmanjom vrednošću.

Primenom metoda zadovoljenja ograničenja dodeliti svakom članu družine sedište u redu, ukoliko se koristi pretraga unapred (*forward checking*) koja se radi nakon dodele vrednosti odabranoj promenljivoj. Za iteraciju precrtati vrednosti koje nisu u domenu svake od promenljivih i zaokružiti vrednosti koje su do tada dodeljene svakoj od promenljivih. Iteracija započinje odabirom sledeće promenljive, a završava se izvršavanjem pretrage unapred.



Zadatak za samostalnu vežbu – Keba Krabin restoran

★★★☆☆

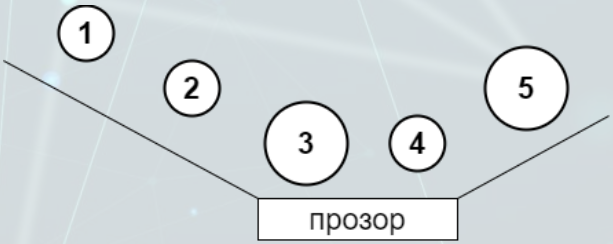
Keba Kraba je vlasnik uspešnog restorana u Koralovu. Kebin najveći motiv jeste novac te mu je u stalnom interesu da što veći broj mušterija bude zadavoljan. Na slici (sa sledećeg slajda) je data skica stolova u restoranu koji su označeni brojevima od 1 do 5. U istom trenutku u restoran ulaze četiri porodice: Ajkulić, Konjić, Račić i Tunić. Sve porodice imaju neke preference pri izboru mesta za sedenje i Keba Kraba kao dobar poznavalac mušterija poznaje njihove želje. Svaka porodica može sedeti samo za jednim stolom i za jednim stolom može sedeti maksimalno jedna porodica. Želje porodica su date u nastavku:

- Porodica Konjić želi da sedi za stolom do prozora (stolovi 3 i 4).
- Porodica Ajkulić želi da sedi za velikim stolom (stolovi 3 i 5).
- Porodice Račić i Konjić su prijatelji i žele da sede za susednim stolovima.
- Porodice Račić i Tunić su prijatelji i žele da sede za susednim stolovima.
- Porodice Ajkulić i Tunić nisu u dobrim odnosima te ne žele da sede za susednim stolovima.

Zadatak za samostalnu vežbu – Keba Krabin restoran

★★★☆☆

Nacrtati graf susednosti, a zatim primenom metoda zadovoljenja ograničenja dodeliti svakoj porodici sto u restoranu, ukoliko se koristi konzistencija luka (*arc consistency*) koja se radi nakon dodele vrednosti odabranoj promenljivoj. Za svaku iteraciju precrtati vrednosti koje nisu u domenu svake od promenljivih i zaokružiti vrednosti koje su do tada dodeljene svakoj od promenljivih. Iteracija započinje odabirom sledeće promenljive ili odabirom nove vrednosti za promenljivu do koje smo došli povratkom unazad (*backtrack*) ili usled nekonzistentnosti luka, a završava se izvršavanjem konzistencije luka.



Zadatak za samostalnu vežbu – Nova godina

★★★☆☆

Jana, jedan od Deda Mrazovih pomoćnika, je bila zadužena da pakuje poklone ove godine. U žurbi je pomešala oznake na poklonima koji se dostavljaju u istu ulicu i sada nije sigurna u koju kuću koji poklon treba da se dostavi. Srećom, Jana se seća nekih informacija o poklonima te je odlučila da pokuša da dodeli svakom poklonu broj kuće. Ukoliko poklone označimo sa A, B, C, D i E, a znamo da su kućni brojevi 1, 2 i 3, tada su informacije sledeće:

- A se dostavlja u kuću čiji je broj veći od broja kuće u koju se dostavlja B.
- Za brojeve kuća u koje se dostavljaju B i C važi: $B + C > 2$.
- D i E se dostavljaju u istu kuću.
- Za brojeve kuća u koje se dostavljaju A i C važi: $A = C + 1$.
- C se dostavlja u kuću čiji je broj manji od broja kuće u koju se dostavlja D.

Zadatak za samostalnu vežbu – Nova godina

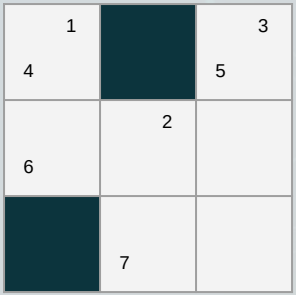
★★★☆☆

Jana je odlučila da reši navedeni problem metodom zadovoljenja ograničenja. Promenljive je odlučila da bira po leksikografskom poretku, dok je vrednosti odlučiča da bira po numeričkom poretku. Prilikom rešavanja problema, Jana koristi i pretragu unapred (forward checking) i proveru konzistentnosti luka (arc consistency). Za svaki od sledećih koraka u algoritmu, u tabeli precrtati vrednosti koje nisu u domenu svake od promenljivih i zaokružiti vrednosti koje su do tada dodeljene svakoj od promenljivih. Svaka iteracija počinje odabirom promenljive i završava se primenom provere konzistentnosti luka. Pre prve iteracije vrši se inicijalna provera konzistentnosti luka.

Zadatak za samostalnu vežbu – Ukrštene reči

★★★★☆

Na slici se nalazi ukrštenica. Relevantna polja su označena odgovarajućim brojevima. Ako je broj u gornjem desnom uglu polja, onda označava početak reči postavljene vertikalno, a ako je broj u donjem levom uglu polja, onda označava početak reči postavljene horizontalno. Uvesti sve potrebne promenljive i ograničenja i primenom metoda zadovoljenja ograničenja popuniti ukrštenicu, ako se koristi pretraga unapred (*forward checking*) uz primenu konzistentnosti lukova (*arc consistency*) nakon dodele vrednosti.



Napisati redosled izbora promenljivih, ako pri izboru biramo promenljivu koja je najviše ograničena (numerički, ukoliko je više takvih), a pri izboru vrednosti promenljivih biramo vrednost koja najmanje ograničava (alfabetski, u slučaju da odabir nije jedinstven).

Ponuđene reči (jedna reč se ne može koristiti više puta):
S, GAS, SE, IBA, R, RAS, BO, OS, AB, ETA, ELO, M, MI, U, UG.

PITANJA?

http://ri4es.etf.rs/