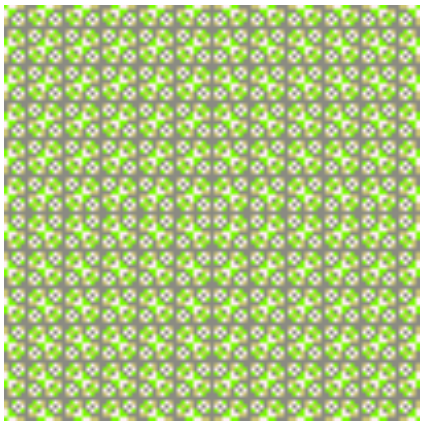
De capul meu prin problema orarului școlar

Vlad Bazon



Momente și schițe de informatică
//docerp.ro

Prefață (cum faci orarul școlii?)

Orarul școlar implică 5 variabile: *obiect* (materiile școlare de parcurs), *profesor* (cei meniți să asigure parcurgerea materiilor), *clasă* (grupuri disjuncte de elevi), *zi* a săptămânii și *oră* din zi; orarul alocă pentru fiecare lecție câte o zi și o oră în care profesorul să intre la una sau alta dintre clasele care i-au fost repartizate, pentru a parcurge la fiecare materia pe care este încadrat.

Profesorii sunt dați, împreună cu clasele (și obiectele) pe care sunt încadrați – variabilele „libere" sunt *ziua* și *ora*, iar acestea trebuie alocate sub condiția principală de a evita suprapunerile (doi profesori nu pot intra simultan, la o aceeași clasă; două clase nu pot avea într-o aceeași oră, un același profesor).

Fiind vorba de două variabile libere, vedem *două* etape: mai întâi distribuim orele profesorilor pe zilele de lucru, apoi distribuim lecțiile din fiecare zi, pe intervalele orare ale zilei; pentru ambele etape, prevedem posibilități de ajustare finală, pentru a satisface eventual, diverse cerințe conjuncturale.

Am adoptat (până la urmă) *limbajul* R, știind totuși că R[1] este un mediu destinat *statisticienilor* – oferind posibilități comode (mai ales, *interactive*) pentru a explora, analiza și sintetiza grafic datele specifice unui domeniu sau proces oarecare. Dar orice limbaj de programare – ca și un instrument oarecare (mai ales dacă are „inteligență artificială") – este mai capabil, decât s-a prevăzut inițial…

Am adoptat R fiindcă în fond, încadrarea profesorilor dintr-o școală dată este un „set de date", care trebuie *completat* cumva, pentru a deveni un orar – care este un alt set de date.
În lucrul obișnuit cu seturi de date (în statistică), o variabilă de tip *factor* are valori cunoscute; de exemplu, în setul rezultatelor elevilor la un examen național, variabila `județ` are valori fixate deja (și permite selectarea ulterioară a datelor unuia sau altuia dintre județe); dar în cazul nostru, factorul `zi` (sau `ora`) are valori chiar necunoscute și sunt multe posibilități pentru fixarea lor într-un moment sau altul, cu îndeplinirea anumitor condiții *ad-hoc*.

Folosind metodele de lucru cu seturi de date, din R (plus „dialectul" modern tidyverse), putem modela concis și chiar eficient, cele două etape menționate mai sus… cu o condiție, am zice: să nu vrem totul deodată!

O fi și frumos, scenariul funcționăresc obișnuit: introduci încadrarea profesorilor (completând formularele puse la dispoziție, pentru fiecare profesor, clasă, etc.) și setezi condițiile dorite, aștepți cuminte orarul final (tabele Excel, HTML sau PDF), îl postezi pe site-ul școlii și gata (…mulțumesc, *aScTimetables*! – face toți banii).

[1] https://www.r-project.org/ *The R Project for Statistical Computing*

Logica noastră este mai laborioasă, dar am zice că este mai deschisă: plecăm de la „tabelul de încadrare” (într-o formulare concisă, simplificând denumirile și implicând poate ceva ingeniozitate, pentru a îngloba coerent și cuplajele de profesori sau de clase); întâi repartizăm lecțiile pe zile (*echilibrat*, și-atât); apoi, pentru fiecare zi, repartizăm lecțiile ei pe orele zilei – fără a lua prea mult în seamă cuplajele și ferestrele; apoi, corectăm suprapunerile ascunse induse de cuplaje, dacă este cazul; apoi, reducem ferestrele apărute, la un număr rezonabil de mic.

„Repartizăm”, „corectăm” și „reducem” înseamnă anumite programe R – în care (ca și în programele de „inteligență artificială”) *întâmplarea* are un rol major (încât obținem rapid fel de fel de „orare” intermediare, între care putem eventual să alegem) – dar implică și mici intervenții interactive (fie direct din consola R, fie chiar și prin anumite aplicații HTML/javaScript).

„Să nu vrem totul deodată” se aplică parcă și limbajului; poți parcurge repede (într-o zi-două) un manual obișnuit, dar „prinzi” limbajul nu ca un scop în sine, ci pe măsură ce îl folosești zi de zi, în contextul judecării coerente a rezolvării unei chestiuni de o anumită complexitate – cum este și *STP*.

În românește nu avem referiri, dar de peste 50 de ani ”*School Timetable Problem*” provoacă și generează numeroase lucrări de doctorat și articole în reviste științifice importante[2]. STP apare – angajând de obicei limbajul C – ca problemă combinatorială de căutare și optimizare, servind concretizării și dezvoltării multor teorii, tehnici și euristici care vor fi adus câștiguri importante pentru matematică, informatică și programare.

Dar dacă întrebi *Cum faci orarul școlii?* – astăzi vei obține ca răspuns nu „folosind calculatorul” sau cine știe ce limbaj și algoritmi, ci doar „folosim *aScTimetables*”…

Software-ul comercial are însă un efect ascuns, de *capcană*: fiindcă acum oricine poate face atâtea și atâtea altele fără să știe cine știe ce (chiar n-ai nevoie să afli de ”STP”, ca să folosești meniurile lui *aScTimetables*), apare ca fiind inutil să mai înveți ceva – „câștigul” devenind cu timpul (nu numai la noi, dar și în „lumea a treia”), profilarea pe *microsoftizare* și dresare funționărească, a învățământului actual; dacă ai de scris un text, fie și un simplu mesaj – deschide și folosește „ca toată lumea” Microsoft-Word (altfel nici nu obții „certificatul de competență digitală”); iar dacă ești programator și ai „de scris” programe – folosește desigur CodeBlocks.

Cam peste tot, STP constă în a repartiza anumite „resurse” (*tupluri* de profesori, grupuri de elevi, săli de clasă) pe anumite „sloturi de timp” – *cupluri* (Zi, Oră), de exemplu `"(Tuesday, 9.00-9.50)"` – astfel încât să se evite suprapunerile și să

[2] A. Schaerf - *A Survey of Automated Timetabling*
http://www.diegm.uniud.it/schaerf/SAS/articles/SurveyTimetabling.pdf

fie satisfăcute într-o proporție cât mai mare, anumite preferințe inițiale.

Noi – plecând nu de la STP, ci mai degrabă de la orare școlare concrete – simplificăm totuși lucrurile: ignorăm resursa „săli de clasă" și decuplăm zilele de lucru, de orele zilei. Pe de altă parte, se clamează iar noi *chiar* considerăm, că orarul școlar trebuie să țină seama în primul rând de anumite principii specifice procesului de învățare în care trebuie angajați elevii (vizând o repartizare *echilibrată*, pe zile, profesori, clase și obiecte); abia apoi, obținând un orar „principial" într-un timp relativ scurt, urmează să se țină seama (dar fără a dezechilibra orarul) și de obișnuitele pretenții individuale.

Precizăm că folosim consola R obișnuită (nu un "IDE", chiar și așa de bun cum este RStudio Desktop); pentru a scrie programele (indiferent de limbaj) folosim un *editor de cod-sursă* (cel obișnuit în Ubuntu-Linux, `gedit`). Această carte a fost scrisă întâi „pe bucăți" în HTML[3], în paralel aproape, cu elaborarea programelor descrise; în final, am transformat (prin pandoc) fișierele HTML respective și am „rescris" (chiar de la capăt, regândind fiecare etapă) în LaTeX.

Respingem recomandarea uzuală de a adăuga un „Capitol introductiv" despre limbajul R: există (chiar dacă nu în română) destule manuale, tutoriale și cărți (v. de exemplu, https://bookdown.org/) privitoare la R, iar simpla comandă `help()` (din consola R a fiecăruia) este chiar suficientă pentru a clarifica *ad-hoc* o funcție sau alta; pe parcurs, facem totuși unele precizări sau sublinieri și asupra limbajului, pentru aspecte mai puțin vizibile (probabil) de pe un manual obișnuit.

În schimb, adăugăm un "P.S." în care prezentăm câteva idei pe care le-am folosit pentru integrarea secvențelor de cod-sursă și a unor tabele de date.

Dacă ar fi să aducem mulțumiri... am zice așa: prețul cărții este 100 lei, din care 80 trebuie îndreptați cu recunoștință către *Free Software Foundation* și vreo 50 autorului; rămâne desigur și un rest consistent (se știe desigur – împărțirea nu este, niciodată, exactă), pentru erori de difuzare.

Vlad Bazon, februarie 2022

[3] http//docerp.ro/ (nov. 2020 - dec. 2021), *Momente și schițe de informatică*

Cuprins

4 Repartizarea lecțiilor pe zilele săptămânii

Cum repartizăm lecțiile, depinde numai de *principii*. Poate că un profesor care are 20 de ore va cere să și le facă în 3 zile: (6, 7, 7, 0, 0); va trebui atunci să înghesuim orele și altor profesori (în alte zile), că vor asta sau nu, iar lecțiile clasei la un obiect sau altul vor fi expediate în două zile. Contează că într-o zi se fac în total 150 de ore, iar într-o alta se fac 200? Contează că la diverse obiecte, orele se fac într-o zi sau două și nu câte una pe zi? Contează că o clasă are într-o zi 4 ore, iar în alta 7 sau 8 ore? Contează și interesele de învățare (și de confort) *ale elevilor*, în toți acești „contează”?

Principiul de bază pe care l-am adoptat este acesta: lecțiile să fie cât mai *uniform* distribuite pe zile, profesori, clase și obiecte; astfel (și pare de la sine înțeles), s-ar asigura condiții propice desfășurării fluente a procesului de învățare în care sunt implicați elevii școlii.

Un algoritm de etichetare

Instrumentăm un algoritm nedeterminist (cu aspecte aleatorii) chiar simplu – bazat pe faptul că *știm* din capul locului că încadrarea dată *admite* repartizări pe zile: însuși planul anual de încadrare, din care am derivat tabelul de încadrare săptămânală, asigură corelațiile necesare asupra numărului de ore pe profesori, discipline și clase; de exemplu, norma orară săptămânală pe profesor este stabilită la 18-27 ore – iar aceasta înseamnă maximum 5-6 ore pe zi; pe clasă, avem în jur de 30 de ore pe săptămână (câte cel mult 6-7 pe anumite obiecte).
Dacă ar exista o *singură* posibilitate de repartizare pe zile a încadrării, atunci ar fi aproape imposibil de a o nimeri, procedând „la întâmplare”; din fericire, sunt posibile foarte multe repartizări pe zile – încât avem suficient de multe șanse de a nimeri una, într-un număr rezonabil de încercări.

Ca ilustrare a ideii de bază, să repartizăm pe zile orele a doi dintre profesori:

```
Zile <- c("Lu", "Ma", "Mi", "Jo", "Vi")  # introdus în stmt_utils.R
twp <- c("p35", "p36")
ls2 <- readRDS("lessons.RDS") %>%  # 979 lecții, prof|cls
       filter(prof %in% twp)  # lecțiile a doi dintre profesori
print(table(ls2$prof)[twp])
#   p35 p36
#    18  17   ## ore/săptămână
D <- cbind(ls2, Zi = rep(Zile, 7))  # etichetează lecțiile, cu zile
```

Pe subsetul celor 18+17=35 de lecții ale celor doi profesori, am adăugat coloana `Zi`, pe care am înscris consecutiv de 7 ori, valorile din `Zile`:

```
   prof cls Zi    12  p35 11C Ma     24  p36 12C Jo
1   p35 10B Lu    13  p35 11C Mi     25  p36 12C Vi
2   p35 10B Ma    14  p35 11C Jo     26  p36 12C Lu
3   p35 10B Mi    15  p35 11C Vi     27  p36  6C Ma
4   p35 10B Jo    16  p35 12A Lu     28  p36  6C Mi
5   p35 10C Vi    17  p35 12A Ma     29  p36  7A Jo
6   p35 10C Lu    18  p35 12A Mi     30  p36  7A Vi
7   p35 10C Ma    19  p35 12A Jo     31  p36  7C Lu
8   p35 10C Mi    20  p35 12A Vi     32  p36  7C Ma
9   p36 10D Jo    21  p36 12A Lu     33  p36  8A Mi
10  p36 10D Vi    22  p36 12A Ma     34  p36  8A Jo
11  p35 11A Lu    23  p36 12A Mi     35  p36  9D Vi
```

Din faptul că liniile profesorului sunt consecutive, rezultă că (prin etichetarea făcută) orele sale sunt distribuite *uniform* (diferind între zile cu cel mult o oră):

```
print(table(D[c('prof','Zi')])[twp, ])
#       Zi
#  prof  Jo Lu Ma Mi Vi
#    p35  3  4  4  4  3
#    p36  4  3  3  3  4
```

Din faptul că între liniile profesorului, cele corespunzătoare unei aceleiași clase sunt consecutive – rezultă că orele profesorului la o aceeași clasă (dacă nu-s mai multe decât 5) se vor desfășura în zile *diferite*.

Dacă în coloana adăugată `Zi` înscriam consecutiv nu secvența `Zile`, ci o permutare oarecare a acesteia – obțineam o altă repartizare posibilă pe zile, diferită (cel mai probabil) de aceea redată mai sus.

Mai departe descriem procedura de etichetare cu zile pe care o vom aplica pentru repartizarea echilibrată a tuturor lecțiilor, pe zilele săptămânii.

Împărțim setul tuturor lecțiilor, *după clasă*; etichetăm succesiv lecțiile unei clase (pentru fiecare clasă, pe rând), astfel: fixăm temporar o ordine oarecare a zilelor de lucru și înscriem secvența respectivă primelor 5 lecții, apoi următoarelor 5, ș.a.m.d. De observat că astfel, clasa va avea în fiecare zi cam *același* număr de ore, iar orele unui aceluiași profesor la clasa respectivă vor fi plasate în zile *diferite* (cu singura excepție când are mai mult de 5 ore, la acea clasă).

Dacă alocarea pe zile făcută astfel clasei *curente*, corelată cu alocările făcute anterior *altor clase*, implică mai mult de 6 ore pe zi la un profesor sau altul – atunci fixăm o altă ordine a zilelor de lucru și *reluăm* etichetarea; la fel, dacă în urma alocării curente, distribuția pe zile a unuia dintre profesori se dezechilibrează.

Dacă etichetarea cu zile nu reușește, pentru niciuna dintre cele 5!=120 de ordonări ale zilelor, atunci schimbăm aleatoriu ordinea profesorilor clasei curente și

reluăm cele maximum 120 de încercări de etichetare cu zile.

Dacă într-o suită de încercări cu câte o altă ordine a profesorilor, nu se reușește etichetarea cu zile pentru clasa curentă – atunci abandonăm *toate* alocările făcute unor clase până la momentul respectiv și reluăm procesul, după ce în prealabil *reordonăm* (aleatoriu) clasele.

Obs. Algoritmul descris mai sus ar trebui cumva amendat pentru cazul când clasele sunt împărțite în două schimburi – caz în care ar fi de dorit ca, pe cât este posibil, profesorilor care au ore în ambele schimburi să li se aloce unele zile pentru primul schimb și altele pentru al doilea. Deocamdată, lăsăm asemenea ajustări în seama unor negocieri *interactive* ulterioare generării prin program a unei repartizări pe zile a lecțiilor.

În exemplul de încadrare considerat aici, toate clasele sunt într-un același schimb; dacă avem două schimburi, le putem trata – pentru a repartiza lecțiile pe zile – fie împreună (ca și cum ar fi un singur schimb), fie pe fiecare schimb în parte.

Anticipând, precizăm că vom folosi un algoritm „de etichetare" similar celui descris mai sus și în cazul repartizării lecțiilor unei aceleiași zile, pe orele zilei (în loc de etichetare cu zile ca mai sus, vom eticheta lecțiile cu orele 1..7 ale zilei).

Lista matricelor de permutări

Vom avea nevoie de *permutări* nu numai pentru a reordona zilele 1..5, dar și mai încolo, pentru a reordona orele 1..7 ale zilei; deci se cuvine să generăm (în prealabil) listele de permutări de `ko`=4..7 elemente.

La CRAN[4] găsim și pachete care conțin funcții de generare a permutărilor (formulate inițial în **C**); de exemplu, `partitions::perms(n)` (care leagă codul compilat al fișierului "`permutation.c`") produce o matrice ale cărei coloane enumeră lexicografic permutările secvenței `1:n`. Însă pachetul respectiv vizează o arie largă de funcții și integrează diverse biblioteci, de care nu avem nevoie; preferăm deci să formulăm direct, o funcție care să ne furnizeze permutările.

Folosim „descompunerea recursivă" obișnuită: 1) elimină elementul curent; 2) procesează *recursiv* restul elementelor; 3) pune înapoi elementul curent (unde „curent" referă pe rând, elementele secvenței inițiale). De exemplu, să obținem o matrice având drept coloane permutări ale secvenței 1:3:

```
> ( M <- cbind(c(2,3), c(3,2)) )
     [,1] [,2]  # permutările mulțimii {2, 3} („curent" = 1)
[1,]    2    3
[2,]    3    2
```

[4] https://cran.r-project.org

```
> Dis1 <- read_csv("re_distr.csv", col_types="cccccc") %>%
          fromRecast()
> saveRDS(Dis1, file = "byDays/CSV/C11v.RDS")
```

Desigur, putem folosi iarăși funcțiile din `test2.R` (`dis_indiv()` și `total_conn()`) pentru a verifica faptul că profesorii conexați prin cuplaje nu cumulează mai mult de 6 ore pe zi, pentru a constata că lecțiile sunt distribuite uniform pe zile (`197` într-o zi și câte `198` în celelalte) și pentru a verifica direct omogenitatea distribuțiilor individuale.

Să observăm că `$prof` este acum (în `C11v.RDS`) de tip *character* și nu *factor*, cum era inițial – dar nu-i cazul să „corectăm”: urmează să repartizăm pe orele zilei, lecțiile fiecărei zile și vom vedea că *este* necesar ca `$prof` să fie „factor ordonat”, însă ordonat pe fiecare zi în parte (și în alt mod decât după numărul de ore).

5 Repartizarea lecțiilor dintr-o zi, pe orele zilei

În subdirectorul convențional `byDays/CSV/` avem deci, repartiția pe zile a tuturor lecțiilor din cursul săptămânii – atât în format CSV specific aplicației `Recast.html`, cât și în format normal `prof|cls|zl`; ulterior, oricând ar apărea necesitatea vreunor modificări, vom putea transfera din nou `re_distr.csv` aplicației ”Recast” pentru a le efectua (reconstituind apoi `C11v.RDS`, corespunzător noilor modificări).

Să aducem distribuția respectivă în directorul de lucru curent, dar sub forma unei liste (sau „dicționar”) care asociază fiecare zi cu lecțiile `prof|cls` repartizate în acea zi (renunțăm la `$zl`, fiindcă are o aceeași valoare pe lecțiile unei aceleiași zile):

```
source("stmt_utils.R")  # tidyverse; Zile
Dzl <- readRDS("byDays/CSV/C11v.RDS")  # 989 lecții prof|cls|zl
L5d <- vector("list")
for(zi in Zile)
    L5d[[zi]] <- Dzl %>% filter(zl == zi) %>%
                 select(prof, cls)  # câmpul 'zl' este de-acum, inutil
saveRDS(L5d, file = "dict_days.RDS")  # zi ==> tabelul prof|cls al zilei
#> str(L5d[1])  ## inspectăm prima dintre cele 5 componente ale listei
# List of 1
# $ Lu:'data.frame':    197 obs. of  2 variables:
#  ..$ prof: chr [1:197] "p10" "p10" "p10" "p10" ...
#  ..$ cls : chr [1:197] "10A" "10B" "10D" "6D" ...
```

Avem de constituit o funcție prin care să alocăm lecțiile repartizate într-o aceeași zi, pe orele 1..7 ale zilei – astfel încât să evităm suprapunerea într-o aceeași oră,

a două lecții (și să evităm formarea prea multor „ferestre”). În principiu, vom aplica un algoritm de etichetare cu orele 1..7 a lecțiilor, analog celui folosit în §4.

Tot „în principiu” (evitând griji suplimentare, pentru a câștiga timp), nu ne va păsa prea mult nici de ferestre, nici de cuplaje: vom avea mai târziu un program prin care să corectăm eventualele suprapuneri apărute la profesorii conexați prin cuplaje și un alt program pentru a reduce cât se poate de mult, numărul ferestrelor.

Ideile și bănuielile de bază

Transformăm `$prof` în *factor ordonat*, după un anumit criteriu – după numărul de ore ale profesorilor pe ziua respectivă, crescător sau descrescător; sau, după vreun alt criteriu, eventual aleatoriu. Împărțim setul tuturor lecțiilor zilei, după clasă; subsetul lecțiilor unei aceleiași clase „moștenește” factorul comun `$prof`, deci lecțiile clasei vor apărea în ordinea stabilită prin `$prof` pentru setul tuturor lecțiilor zilei.

Pentru fiecare clasă, una după alta, încercăm să alocăm lecțiile acesteia pe orele `1..ko` ale zilei, `ko`=4..7 fiind numărul lecțiilor clasei, în ziua respectivă; cu alte cuvinte, anexăm subsetului de lecții ale acelei clase, o nouă coloană, `$ora`, pe care înscriem secvența `1..ko` (sau una dintre permutările acesteia).

La un moment dat, se poate ca alocarea încercată pentru clasa curentă să eșueze – venind în conflict cu alocarea pe ore făcută anterior unei alte clase (un profesor ar intra într-o *aceeași* oră la ambele clase); dacă oricum am permuta secvența `1..ko` din coloana `$ora`, nu putem evita conflictul cu alocările făcute anterior altor clase – atunci stopăm procesul, reordonăm clasele (după un criteriu sau altul) și apoi reluăm pentru noua ordine a claselor, încercările de alocare pe ore a lecțiilor fiecăreia.

Este de bănuit că dacă orele 1..7 sunt montate fiecărei clase începând cu profesorii care au cel mai multe ore în ziua respectivă, atunci vor rămâne din ce în ce mai puține „portițe” pentru lecțiile profesorilor de la sfârșit (care, având puține ore în acea zi, apar la cel mult trei clase) – încât vor fi necesare foarte multe reveniri, până ce se va nimeri o ordine a claselor în care să nu mai apară conflicte între alocările succesive ale lecțiilor claselor, pe orele zilei.

Cu alte cuvinte, pare preferabil să ordonăm lecțiile *crescător* și nu descrescător, după numărul de ore din acea zi ale profesorilor; nu numai că s-ar reduce numărul de reveniri, dar este de bănuit că și numărul ferestrelor ar deveni mai mic: un profesor care are la clasa respectivă 5 sau mai multe ore nu va putea avea mai mult de una sau două ferestre, pe când dacă are 2 sau 3 ore, va putea avea chiar și 5 sau 4 ferestre – deci făcând alocarea în ordinea *crescătoare* a numărului de ore

ale profesorilor este de așteptat ca majoritatea profesorilor care au puține ore, să le aibă așezate compact (fără ferestre) în cadrul zilei.

Dar ordonarea după numărul de ore este insuficient de relevantă, fiindcă sunt multe cazuri de profesori care au un *același* număr de ore, în ziua respectivă; pe de altă parte, trebuie să conteze nu numai numărul de ore, dar și clasele asociate orelor respective: unii profesori au mai multe clase în comun, alții – mai puține. S-ar cuveni să ordonăm profesorii (și deci, lecțiile) și după acest criteriu: au sau nu, clase în comun. Posibilitățile de plasare a orelor depind cumva, de câte clase au în comun unii și alții: sunt mai multe, dacă au mai multe clase în comun – deci pare preferabil să începem alocarea orelor cu aceia care au un număr mic de clase în comun și să lăsăm mai la urmă, pe cei cu multe clase comune.

Fiindcă pentru alocarea pe orele zilei a lecțiilor acelei zile, parcurgem clasele pe rând (într-o anumită ordine a claselor) – este de bănuit în final, că cel mai bine ar fi ca profesorii (deci și lecțiile de etichetat cu ore, la fiecare clasă) să fie ordonați astfel încât *fiecare să aibă cât mai puține clase în comun, cu cei care îl preced.*

Totodată, chiar dacă parcurgem clasele într-o ordine aleatorie, unele clase ar trebui să fie cumva favorizate; de exemplu, o clasă care are 4 ore (cu 4 profesori) ar fi de abordat înaintea celor cu 6 ore (6 profesori) – fiindcă altfel, întâlnind-o mai spre sfârșit, cel mai adesea profesorii acelei clase ar avea deja ocupate primele 4 ore. Cu alte cuvinte, ordinea de parcurgere a claselor ar trebui să reflecte un principiu analog celui conturat mai sus pentru profesori: *fiecare să aibă cât mai puțini profesori comuni cu clasele precedente.*

Precizăm că bănuielile de bun-simț strecurate mai sus au fost verificate experimental, pe un număr suficient de mare de cazuri. Am constatat că ordonând descrescător după numărul de ore, ajungem (pe calea evidențiată mai sus) la un orar al zilei cu mai multe ferestre și într-un timp mai lung, decât dacă am ordona crescător; apoi, am constatat că ordonând crescător după numărul de clase comune și respectiv, ținând cont de numărul de profesori comuni – ajungem cel mai repede la orar, cu mai puține ferestre decât în celelalte cazuri.

Coeficienții *betweenness*, pentru profesori și clase

Încercând să formalizăm bănuiala conturată mai sus – asupra poziționării fiecărui profesor în raport cu ceilalți, condiționată de prezența unor clase comune (și asupra unei ordonări a claselor care să țină seama cumva, de numărul de profesori comuni) – ajungem la măsura numită betweenness (întâlnită de exemplu, în studiile statistice privitoare la rețelele de socializare).

Considerăm un *graf* Γ având drept vârfuri profesorii existenți în ziua pentru

care vrem să obținem un orar, două vârfuri fiind adiacente (legate printr-o muchie a grafului) dacă profesorii respectivi au măcar o clasă comună.

Am vrea să parcurgem cât mai „ușor" acest graf, trecând deci de la un vârf la următorul pe un cel mai scurt – *geodezică* – dintre drumurile existente în Γ între cele două vârfuri. Considerând arbitrar două vârfuri, P1 și P2, probabilitatea ca un anumit vârf P să fie „între" cele două date – adică să se afle pe o geodezică de la P1 la P2 – este dată de raportul dintre numărul geodezicelor de la P1 la P2 care trec prin P și numărul tuturor geodezicelor de la P1 la P2; valoarea acestui raport este „notată" `betweenness(P1, P, P2)`, iar însumând toate aceste valori când P1 și P2 variază în mulțimea tuturor vârfurilor – obținem **`betweenness`**`(P)`, pentru fiecare vârf P al grafului.

Bineînțeles că nu este necesar să modelăm noi, funcția `betweenness()`; putem folosi de exemplu, pachetul **`igraph`**:

```
# between.R (coeficienții 'betweenness' pentru profesori și clase)
source("stmt_utils.R")
library(igraph)  # graph_from_adjacency_matrix(); betweenness(); etc.
load("Messing.Rda")  # conexiunile săptămânale induse de cuplaje
```

Întâi, prevedem o funcție prin care să obținem un „dicționar" necesar ulterior pentru a defini prin matricea de adiacență, graful profesorilor și respectiv, graful claselor: se asociază profesorilor câte un vector conținând clasele fiecăruia, respectiv se asociază claselor câte un vector conținând profesorii fiecăreia.

Desigur, putem formula câte o funcție pentru fiecare dicționar – dar cele două funcții ar diferi *doar* prin comutarea între ele a variabilelor `prof` și `cls`: pentru fiecare profesor (respectiv, clasă) se decupează lecțiile zilei care au în coloana `prof` (respectiv, `cls`) acel profesor (respectiv, acea clasă) și din subsetul obținut se extrag valorile distincte din coloana `cls` (respectiv, `prof`).

Însă putem obține dicționarul respectiv și într-un caz și în celălalt, printr-o *aceeași* funcție – folosind mecanismul NSE (*non-standard evaluation*) instituit de tidyverse, bazat pe funcții ca `enquo()` și pe operatorul '!!':

```
dict_col <- function(LSS, col) { # col: variabila prof, sau cls
    col1 <- enquo(col)  # "quosure" care, în contextul LSS,
                        # va reda valorile din coloana referită
    col2 <- quo(setdiff(colnames(LSS),
                        as_label(col1)))  # cealaltă coloană
    FxS <- LSS %>% distinct(!!col1) %>%
                   pull()  # !! dă conținutul variabilei referite
    SxF <- map(FxS, function(X)
               LSS %>% filter(!!col1 == X) %>%
```

```
                select(!!col2) %>% distinct() %>% pull())
    names(SxF) <- FxS
    SxF  # dicționar profesor ==> vectorul claselor proprii, sau
         # dicționar clasă ==> vectorul profesorilor clasei
}
```

`dict_col()` trebuie apelată furnizând ca argument '`col`' fie `prof` (*fără* ghilimele), fie `cls` – variabile (*nu* șiruri de caractere) care referă câte una dintre coloanele setului de lecții `LSS` (de altfel, în termeni specifici explorării și analizei datelor, nici nu se zice „coloane", ci chiar „variabile" și se zice nu „linii", ci „observații" asupra variabilelor). În corpul funcției, `enquo(col)` asociază variabilei `col` primită ca argument, o anumită expresie funcțională (*quosure*) care atunci când va fi citată sub operatorul '`!!`' într-un context în care există `LSS` (de exemplu, după "`LSS %>%`"), va produce valorile din coloana respectivă a lui `LSS`.
Următoarea exemplificare este instructivă și dintr-un alt punct de vedere:

```
> for(Q in c("11D", "11E")) print(dict_col(LSS, cls)[[Q]])
[1] "p39" "p22p17" "p24" "p21" "p45"  # la 11D
[1] "p10" "p07" "p24" "p21" "p51" "p54"  # la 11E
```

Observăm că `p39` și `p51` apar numai la câte una dintre clasele `11D` și `11E`; ori cei doi trebuie considerați ca având măcar o clasă comună (cea desemnată prin "11DE"). Deocamdată ignorăm această situație (vom trata separat, cazul claselor cuplate – analog tratării de la repartizarea lecțiilor pe zile); dar, având în vedere că avem destule cuplaje, va trebui să luăm măsuri astfel încât profesorii fictivi să fie adiacenți măcar celor pe care îi cuplează direct (altfel, coeficienții "betweenness" ai lor vor fi mai toți `0`, ori egalitățile nu sunt de dorit pentru ordonare).

În schimb, pentru graful claselor n-ar fi nevoie să fim așa de exigenți – este suficient să considerăm adiacente clase care au măcar un profesor comun (ignorând situația când nu au profesori comuni, dar într-una apare un profesor fictiv, iar în cealaltă un profesor conexat prin cuplaje cu acesta – caz în care ar trebui totuși, să forțăm adiacența între cele două clase). Clasele vor fi angajate mereu, într-o ordine aleatorie „cu preferințe", iar adăugarea câtorva adiacențe particulare nu va influența, cât de cât semnificativ, ordinea necesară.

Graful claselor poate fi construit mai ușor:

```
graph_cls <- function(LSS) {
    lcp <- dict_col(LSS, cls)
    Cls <- names(lcp)
    len <- length(Cls)
    adjm <- matrix(rep(0, len), nrow=len, ncol=len, byrow=TRUE,
                   dimnames = list(Cls, Cls))  # matricea de adiacență
```

```
    adj2cls <- function(K1, K2)  # adiacente dacă au profesori comuni
               length(intersect(lcp[[K1]], lcp[[K2]])) > 0
    for(K1 in Cls)
        for(K2 in Cls)
            if(K1 != K2)
                adjm[K1, K2] <- adj2cls(K1, K2)
    graph_from_adjacency_matrix(adjm, mode = "undirected")
}
```

Analog avem pentru graful profesorilor, dar formularea adiacenței este mai pretențioasă (aici o expediem cu '||' și '&&'):

```
graph_prof <- function(LSS) {
    lcp <- dict_col(LSS, prof)
    Prof <- names(lcp)
    len <- length(Prof)
    adjm <- matrix(rep(0, len), nrow=len, ncol=len, byrow=TRUE,
                   dimnames = list(Prof, Prof))  # matricea de adiacență
    adj2prof <- function(P1, P2) {
        if(length(intersect(lcp[[P1]], lcp[[P2]])) > 0) return(1L)
        if(P1 %in% names(Tw1) && (P2 %in% Tw1[[P1]]
                                  P2 %in% Tw2[Tw1[[P1]]])
           P2 %in% names(Tw1) && (P1 %in% Tw1[[P2]]
                                  P1 %in% Tw2[Tw1[[P2]]])
           P1 %in% names(Tw2) && P2 %in% Tw2[[P1]]
           P2 %in% names(Tw2) && P1 %in% Tw2[[P2]])  return(1L)
        0L
    }  # adiacenți dacă au clasă comună, sau sunt conexați prin cuplaje
    for(P1 in Prof)
        for(P2 in Prof)
            if(P1 != P2)  adjm[P1, P2] <- adj2prof(P1, P2)
    graph_from_adjacency_matrix(adjm, mode = "undirected")
}
```

Pentru grafurile returnate de aceste funcții – pe seturile zilnice de lecții rezultate anterior în `dict_days.RDS` – vrem coeficienții "betweenness" ai vârfurilor:

```
Z <- readRDS("dict_days.RDS")  # dicționarul lecțiilor pe fiecare zi
BTW_prof <- vector("list")
BTW_cls <- vector("list")
for(zi in Zile) {
    LSS <- Z[[zi]]  # lecțiile prof|cls din ziua curentă
    g1 <- graph_prof(LSS)
    BTW_prof[[zi]] <- sort(betweenness(g1, directed = FALSE))
    g2 <- graph_cls(LSS)
```

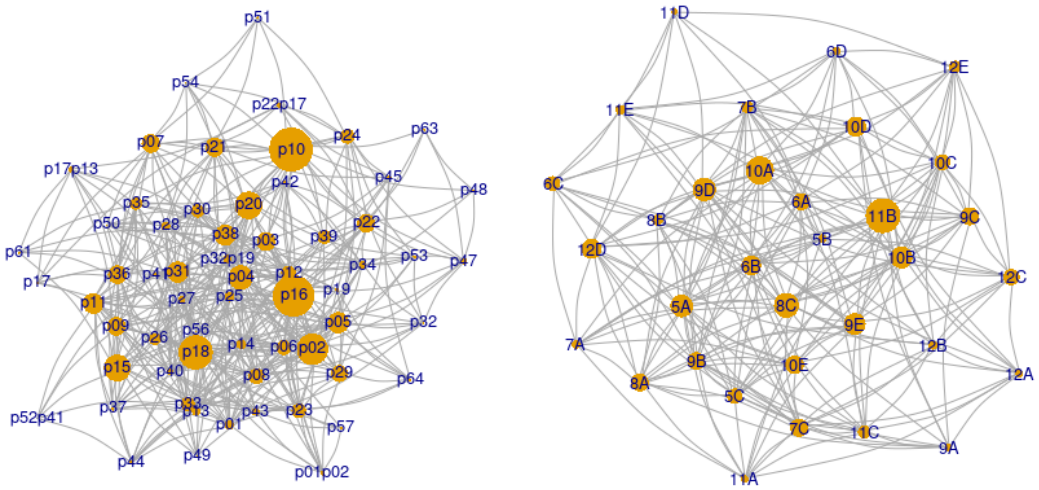
```
    BTW_cls[[zi]] <- sort(betweenness(g2, directed = FALSE))
}
save(BTW_prof, BTW_cls, file = "BTW.rda")
```

Când ne vom ocupa mai încolo, de repartizarea pe orele zilei a lecțiilor dintr-o aceeași zi, pentru toate zilele – vom recupera prin `load("BTW.rda")` cele două liste ai vectorilor de coeficienți "betweenness" ai profesorilor și claselor pentru fiecare zi și vom transforma variabila `prof` a lecțiilor zilei curente în *factor* ordonat după valorile din `BTW_prof[[zi]]`, iar ordinea aleatorie în care vom parcurge clasele va fi ponderată cumva prin valorile `BTW_cls[[zi]]`.

Desigur, în loc de a reda aici vectorii respectivi, putem produce grafic (prin `plot()`) grafurile respective, sugerând valorile "betweenness" prin proprietatea `$size` asociată vârfurilor grafului și folosind una sau alta dintre funcțiile igraph::`layout_...()`:

```
plot_btw <- function(zi) {
    LSS <- Z[[zi]]  # lecţiile din ziua curentă
    g1 <- graph_prof(LSS)
    btw1 <- BTW_prof[[zi]]
    g2 <- graph_cls(LSS)
    btw2 <- BTW_cls[[zi]]
    rord1 <- map_dbl(V(g1), function(v)
                     which(names(btw1) == names(V(g1)[v])))
    rord2 <- map_dbl(V(g2), function(v)
                     which(names(btw2) == names(V(g2)[v])))
    g1 <- permute(g1, rord1)  # vârfurile - în ordinea coeficienţilor!
    g2 <- permute(g2, rord2)
    par(mfrow = c(1, 2), mar=c(1.5, 3, 1.5, 2))
    for(G in list(g1, g2)) {
        btw <- if(gorder(G) > 40) 0.2*btw1 else 0.75*btw2
        coords <- layout_with_lgl(G)
        plot(G,
            vertex.size = btw,  # proporţionează mărimea după coeficienţi
            vertex.frame.color=NA,
            vertex.label.family="sans",
            edge.curved = .2,
            layout = coords,
            margin = rep(-0.25, 4),
            vertex.label.cex = 0.8
        )
    }
}
```

```
plot_btw(1)  # grafurile pentru ziua "Lu"
```



Subliniem că odată create, cele două obiecte *igraph* au vârfurile (care se pot obține prin funcția `V(G)`) într-o ordine fixată (prin indecși numerici `1..gorder(G)`) și am folosit `permute()`, pentru a reindexa vârfurile după coeficienții de "betweenness" – putând astfel să le redăm grafic cu o mărime proporțională cu aceștia.

Pe imaginea rezultată vedem de exemplu că pentru ziua `Lu`, cele mai mari valori de "betweenness" corespund profesorilor `p10` și `p16` respectiv, claselor `11B` și `10A`; iar `p54`, `p52p41`, `p17` etc., respectiv `11D`, `9A` etc. se află pe un număr mult mai mic de geodezice ale grafurilor respective – încât, amânarea alocării orelor pentru aceste clase diminuează mult posibilitățile, mărind riscul ca, ajungând aproape de sfârșit, să trebuiască să reluăm de la capăt (după ce reordonăm clasele) procesul de alocare clasă după clasă, a orelor.

Ordonarea aleatorie preferențială

Procesul de alocare pe orele zilei a lecțiilor din acea zi începe cu lecțiile uneia dintre clase, continuă cu lecțiile unei alte clase, ș.a.m.d. urmărind mereu să nu apară suprapuneri; dacă la un anumit moment, lecțiile clasei curente nu mai pot fi plasate pe orele zilei, apărând mereu suprapuneri cu alocările făcute anterior altor clase – atunci pentru a finaliza totuși alocarea tuturor lecțiilor zilei, avem două posibilități: folosim metoda binecunoscută *backtracking*, respectiv cum vom face noi (în loc de a reveni mereu asupra unor alocări făcute anterior, modificându-le și reluând astfel până ce reușim să trecem de clasa curentă): *stopăm* procesul de alocare în curs, *reordonăm* cumva lista claselor și *relansăm* procesul de alocare, abordând rând pe rând clasele conform noii ordini a acestora.

Lecțiile săptămânii au fost astfel repartizate pe zile, încât putem fi siguri că *există*

mijlocul tabelului) se regăsesc deasupra de mai puține ori, decât dedesubtul liniei; în general, două clase de pe o aceeași linie, dacă apar și deasupra liniei, apar la profesori diferiți.

Nu dăm importanță nici ferestrelor (despre care observăm că la un profesor sau altul, sunt cel mult două – cum ceream în `mount_hours()`), nici faptului că un profesor sau altul are orele în prima parte a zilei (mai ales dacă are mai puțin de 4 ore, fiindcă atunci are "betweenness" mic), sau în a doua: urmează să modificăm orarul, prin alte programe.

Să observăm că (*întâmplător*, fiindcă în program n-am luat nicio măsură pentru aceasta) `p39` și `p51` au clasele `11D` și respectiv `11E` într-o aceeași oră (a 5-a) – astfel că partajul de clase "11DE" este deja asigurat.

Fiindcă în `mount_hours()` am ignorat cuplajele, au apărut situații de *suprapunere* de ore, la profesori conexați prin cuplajele respective; de exemplu, vedem în tabelul de mai sus că profesorul fictiv `p01p02` (pe linia 21) are orele 1 și 2 la clasele `9C` și `9A` – însemnând că în aceste ore trebuie să fie liber și `p01` (linia 32) și `p02` (linia 59); însă `p01` este ocupat în orele 1 și 2, având ore la `7B` și `9E`.
La fel, pentru încă un exemplu de suprapunere, `p13` are ora 2 la `8A`, dar tot ora 2 trebuie să intre împreună cu `p17` la clasa `12A`.

Să mai observăm că ignorând cuplajele, numărul total de ferestre considerat mai sus este doar *estimativ*: pot exista și ferestre „ascunse", la profesori angajați în mai multe cuplaje; de exemplu, `p22` (linia 46) are orele proprii 4-6 (fără ferestre), dar mai are ora 2 împreună cu `p17` (linia 22) – deci are o fereastră în ora a 3-a.

6 Matricele orare; mutări Kempe

Am obținut repede „orarul", dar iată că el trebuie corectat – fiindcă pe de o parte, există unele suprapuneri ascunse, la profesori angajați în cuplaje; pe de altă parte, există prea multe ferestre. În plus, avem de menținut într-o aceeași oră, clasele cuplate (precum "11DE").

Cum am putea corecta? În principiu, trebuie să mutăm o lecție `prof`|`cls` – sau… o lecție `cls`|`prof` – dintr-o oră a zilei într-o altă oră a zilei.
De exemplu, să observăm liniile 31 și 56 din orarul redat mai sus:

	prof	*`1`*	*`2`*	*`3`*	*`4`*	*`5`*	*`6`*	*`7`*
31	p28	–	10E	7A	12D	5A	–	–
56	p04	8A	–	6A	5C	12D	10E	–

Pentru a elimina fereastra din ora 2 a lui `p04`, putem muta (în cadrul liniei 56) clasa `10E` din coloana 6, în coloana 2; însă la `10E` fusese fixată deja o lecție în

ora 2 – anume, cu `p28` – deci pentru a evita suprapunerea, trebuie să facem și o mutare inversă: pe lina 31, mutăm `10E` din coloana 2 în coloana 6. După această pereche de mutări nu mai apar suprapuneri (nici „ascunse”, profesorii respectivi nefiind implicați în cuplaje, în ziua `Lu`) și avem cu o fereastră mai puțin.

Să observăm că o „mutare” angajează doar *coloanele* orare; era suficient să zicem „mută `10E` din coloana 6 în coloana 2”, fiindcă pe coloane clasa nu poate apărea decât o singură dată (deci nu este necesară precizarea liniei pe care se face mutarea). Deși vorbim de mutări de lecții, coloana `$prof` joacă doar un rol informativ, etichetând suplimentar liniile (referite altfel implicit, prin „numărul curent”).

Funcțiile următoare (de plasat în `stmt_utils.R`) transformă un orar `prof|cls|ora` în „matrice orară” *după profesori*, respectiv *după clase*:

```
prof_matrix <- function(X) {  # orarul prof|cls|ora al unei zile
    Prof <- levels(X$prof)
    M <- as.matrix(orarByProf(X))  # profesor → clase
    M <- M[, 2:ncol(M)]
    M[is.na(M)] <- '-'
    row.names(M) <- Prof
    M  # "matricea orară" după profesori
}
cls_matrix <- function(X) {  # orarul prof|cls|ora al unei zile
    Cls <- unique(X$cls)
    M <- as.matrix(orarByCls(X))  # clasă → profesori
    M <- M[, 2:ncol(M)]
    M[is.na(M)] <- '-'  # oră liberă, sau fereastră
    row.names(M) <- Cls
    M  # "matricea orară" după clase
}
```

În matricele orare, profesorii și respectiv clasele, devin „*nume* de linii”; prima va servi pentru a „repara” ferestre, prin mutări de clase, iar a doua matrice – pentru a corecta suprapunerile ascunse, prin mutări de profesori dintr-o coloană în alta:

```
> Z <- readRDS("TMT_days1.RDS"); Lu <- Z[[1]]
> pm <- prof_matrix(Lu)  # pentru a corecta ferestre, mutând clase
> print.table(pm[c("p28", "p04"), ])
#       1   2   3   4  5   6  7
#    p28 -  10E 7A 12D 5A  -   -
#    p04 8A -   6A 5C  12D 10E -
> cm <- cls_matrix(Lu)  # pentru a corecta suprapuneri, mutând profesori
> print.table(tail(cm, 3))
#       1      2   3   4   5   6   7
#    9C p01p02 p23 p12 p16 p29 p06 -
```

```
#    9B p32p19 p40 p26 p18 p21 p36 -
#    9C p40    p01 p14 p06 p09 p05 -
```

Subliniem iar, că (în mod normal) în matricele orare fiecare element (clasă, respectiv profesor) apare cel mult o singură dată, pe oricare coloană (altfel, avem „suprapunere” – interzisă într-un orar corect); deci *coloana* joacă rolul major, pentru referirea unui element al matricei (nu linia, pe care pot fi și elemente egale).

Indecșii de coloană ai unui profesor

O matrice este în fond un *vector*, atributat (prin `dim()`) cu două „dimensiuni”:

```
> (al <- LETTERS[1:12])  # un vector, cu primele 12 litere
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
> dim(al) <- c(3, 4); al  # ca matrice cu 3 linii şi 4 coloane
     [,1] [,2] [,3] [,4]
[1,] "A"  "D"  "G"  "J"  # vectorul este mapat pe coloane succesive
[2,] "B"  "E"  "H"  "K"
[3,] "C"  "F"  "I"  "L"
> which(al %in% c("E", "I"))  # indecşi de elemente ale vectorului
[1] 5 9
> al[5]  # al 5-lea element al vectorului subiacent matricei
[1] "E"
> al[c(5, 9)]
[1] "E" "I"
```

Pentru niște elemente indicate, `which()` ne dă indecșii aparițiilor acestora în cadrul *vectorului* subiacent matricei respective. Dar pentru „matricele orare”, *coloanele* joacă rolul important: indică ora 1..7 din zi, alocată lecției; pe linie se poate repeta un profesor sau o clasă, însă pe o coloană fiecare profesor sau clasă trebuie să apară cel mult o singură dată. Deci ne interesează nu atât indecșii returnați de `which()`, cât *indecșii de coloană*, ai lecțiilor respective.

Următoarea funcție – introdusă în `stmt_utils.R` – convertește indecșii returnați de `which()` pentru un element dat al matricei (indecși „absoluți”, în vectorul unidimensional subiacent matricei), în indecși de coloană (folosind '`%/%`' pentru împărțirea întreagă "DIV" și '`%%`' pentru "MOD"):

```
which_cols <- function(M, elm) {
    nrw <- nrow(M)  # numărul de valori dintr-o coloană a matricei
    map_int(which(M %in% elm), function(idx)
            ifelse(idx %% nrw == 0L, idx %/% nrw, idx %/% nrw + 1L))
}  # coloanele care conţin cel puţin o dată, elementul indicat
```

Poate că puteam folosi `which(M == elm)`, în loc de `which(M %in% elm)`; efectul

este același, dar între cei doi operatori avem totuși o deosebire importantă: `==` *păstrează* atributele obiectului pe care se face căutarea, pe când `%in%` returnează totdeauna un vector (unidimensional):

```
> str(M == c("p01", "p32"))  # M este matricea orară a unei zile
 logi [1:28, 1:7] FALSE TRUE FALSE FALSE ...  # matrice logică [28, 7]
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:28] "10A" "10B" "10C" "10D" ...
  ..$ : chr [1:7] "1" "2" "3" "4" ...
> str(M %in% c("p01", "p32"))
 logi [1:196] FALSE TRUE FALSE FALSE ...  # vector logic [196]
```

Introducând „matricea orară” și „indecșii lecțiilor”, deja am făcut un pas important – acum putem depista ușor (deocamdată interactiv) suprapunerile ascunse:

```
> lu <- cls_matrix(Z[["Lu"]])
> prc <- c("p01p02", "p01", "p02")  # profesori conexați prin cuplaje
> ore <- lapply(prc, function(P) which_cols(lu, P))
> names(ore) <- prc; print(ore)
$p01p02  [1] 1 2  # p01p02 apare în coloanele (orele) 1 și 2
$p01     [1] 1 2 3 6
$p02     [1] 4 5 6 7  # p02 apare în coloanele 4-7
```

Am considerat un vector cu profesori conexați prin cuplaje – din `mount_hours()`, numai între aceștia puteau să apară suprapuneri – și aplicându-i `which_cols()` (prin `lapply()`), am obținut indecșii coloanelor în care se află fiecare; se vede imediat că avem o suprapunere, în coloana 2, anume la `p01p02` și `p01` (și am putea corecta, mutând `p01` din coloana 2 în coloana 4, de exemplu).

Mai trebuie să lămurim principiul după care, într-o matrice orară, mutăm în altă coloană o clasă, respectiv un profesor (și ar fi al doilea „pas important”).

Lanțuri Kempe, în graful lecțiilor din două coloane

Mutarea în altă coloană implică un lanț de schimburi succesive între cele două coloane, pentru a evita apariția unor suprapuneri; pentru a vedea despre ce este vorba (încât ulterior, să formulăm funcțiile de mutare necesare), să considerăm graful lecțiilor din cele două coloane – de exemplu, pentru matricea orară pe profesori.

Să desprindem din orar lecțiile repartizate în două coloane oarecare – să zicem, cele corespunzătoare orelor 3 și 5 ale zilei; dar vorbim de „lecții”, deci ignorăm liniile libere ”`- -`”; ignorăm deasemenea, lecțiile identice (de exemplu ”`10E 10E`”), precum și cazurile banale – adică perechile de linii de genul ”`5C -`” și ”`- 5C`”, pentru care mutarea clasei este „imediată”:

```
# kempe_chain.R
source("stmt_utils.R")  # prof_matrix()
Z <- readRDS("TMT_days1.RDS")
Lu <- Z[["Lu"]]
Mp <- prof_matrix(Lu)
h45 <- Mp[, c(3, 5)]  # lecțiile repartizate în orele 3 și 5
EDG <- h45[ h45[, 1] != h45[, 2], ]  # exlude "- -", sau "10E 10E"
colnames(EDG) <- c("q1", "q2")
#print.table(EDG)
exc <- vector("integer")
for(i in 1:nrow(EDG)) {  # exclude perechi ca ("5C -"), ("-", "5C")
    if(EDG[i, 2] == '-') {
        for(j in 1:nrow(EDG)) {
            if(EDG[j, 1] == '-' & EDG[j, 2] == EDG[i, 1]) {
                exc <- c(exc, i, j)
            }
        }
    }
}
EDG <- EDG[-exc, ]  # setul arcelor (ora3 --> ora5)
```

Pentru a muta o clasă ”K” din coloana `q1` în coloana `q2` (fără a avea în final, suprapuneri), trebuie să înlănțuim cumva, linia pe care avem ”K”, cu liniile din `EDG` în care clasa inițială este cea finală de pe linia precedentă – până regăsim clasa ”K”:

```
  q1  q2
  K   x1  # primul schimb între coloane: K <--> x1 (rezultă: x1 K)
  x1  x2  # evită suprapunerea:         x1 <--> x2 (rezultă: x2 x1)
  x2  x3
  x3  K     # rezultă un circuit: K → x1 → x2 → x3 → K
```

Deci secvența de mutări care asigură schimbarea unei clase dintr-o coloană în alta, formează un *circuit* (drum închis), în graful care are drept arce `q1 -> q2`, unde `(q1 q2)` este o linie din setul `EDG`.

Din setul de arce `EDG` obținem un obiect de tip `IGRAPH`:

```
require(igraph)
g35 <- graph_from_edgelist(EDG)
```

```
# IGRAPH 51938cb DN-- 30 37 --  # graf orientat; 30 vârfuri, 37 arce
# + attr: name (v/c), color (v/c)
# + edges from 51938cb (vertex names):
# [1] 10E->-   6D ->-   -  ->9C  11A->-   12D->-   10B->-   6C ->-   6B ->-
# [9] 9A ->5A  12E->12D 12A->-   -  ->11B 5A ->11A 5C ->10B 7C ->8B  9D ->7A
#[17] 6A ->6B  9E ->9B  -  ->12C 12B->12A -  ->5C  5B ->10E -  ->10C 8B ->8A
```

```
#[25] 10A->11C 12C->10A -  ->12E 7A ->7C  9B ->7B  9C ->5B  -  ->9E  8A ->6C
#[33] 7B ->6D  -  ->9D  11C->12B 11B->9A  10C->6A
```

Pentru a găsi circuitele lui `g35` (care este un graf „mic” – pe ecran au încăput toate arcele), putem aplica „parcurgerea în adâncime” (*depth-first search*), speculând funcția `igraph::dfs()`. Alegem ca nod de plecare `root`, vârful '–' – fiindcă acesta apare în mai multe poziții pe fiecare dintre cele două coloane din `EDG` (spre deosebire de clasele propriu-zise, care apar fiecare câte o singură dată), deci figurează într-un număr mare de circuite ale lui `g35` (dar nu neapărat, în toate):

```
DFS <- dfs(g35, root='-', neimode="out", order.out=TRUE, dist=TRUE)
```

`dfs()` returnează o serie de date, conform parametrilor impliciți (de exemplu, `order=TRUE`) și celor explicitați în linia de apel:

```
$order  + 30/30 vertices, named, from 51938cb:
 [1] -   9C  5B  10E 12E 12D 11B 9A  5A  11A 5C  10B 9D  7A  7C  8B
[17] 8A  6C  9E  9B  7B  6D  12C 10A 11C 12B 12A 10C 6A  6B

$order.out  + 30/30 vertices, named, from 51938cb:
 [1] 10E 5B  9C  12D 12E 11A 5A  9A  11B 10B 5C  6C  8A  8B  7C  7A
[17]  9D  6D  7B 9B  9E  12A 12B 11C 10A 12C 6B  6A  10C -

$dist
10E   -  6D  9C 11A 12D 10B  6C  6B  9A  5A 12E 12A 11B  5C  7C  8B
  3   0   4   1   4   2   2   6   3   2   3   1   5   1   1   3   4
 9D  7A  6A  9E  9B 12C 12B  5B 10C  8A 10A 11C  7B
  1   2   2   1   2   1   4   2   1   5   2   3   3
```

`DFS$order` conține vârfurile grafului în ordinea vizitării acestora; `DFS$order.out` conține vârfurile în ordinea revenirii pe ramura curent parcursă (de la frunză la nodurile precedente, pentru a descoperi vreo altă ramificare din acestea). `DFS$dist` este un „dicționar” care precizează pentru fiecare vârf, distanța lui față de rădăcina arborelui de căutare (numărul de arce ale drumului de la rădăcină la vârf).

Căutarea pleacă din '–' – *primul* nod din `$order`; fiindcă pe traseul invers `$order.out`, nodul '–' apare (în acest caz) la sfârșit – deducem că *toate* nodurile lui `G` au putut fi atinse plecând din '–' (altfel spus, `g35` este un graf *conex*).

Nodul `9C` (al doilea din `$order`) are `dist['9C']=1`, iar următorul nod la distanța 1 față de '–' este `12E`; deducem că nodurile aflate în `$order` înaintea lui `12E`, aparțin aceluiași circuit: '–' ⟶ `9C` ⟶ `5B` ⟶ `10E` ⟶ '–'.
Analog putem găsi mai departe, celelate circuite din `g35`.

Dacă graful nu este conex, există un vârf care *nu* se află pe niciunul dintre drumurile care trec prin '–'; atunci `dfs()` – câtă vreme nu schimbăm parametrul

implicit *unreachable*=TRUE – va fi reluat automat, dintr-un asemenea vârf, rezultând și circuitele care nu trec prin '–'.
De exemplu, pentru graful g12 al lecțiilor din primele două coloane, găsim:

```
$order  + 30/30 vertices, named, from 82f0a81:
 [1] -   5B  9C  11E 12D 7A  10D 6A  8C  11D 8B  6C  6B  7B  9E  9B
[17] 9D  10E 12B 12C 7C  8A  5C  9A  11C 12A 5A  10C 11B 11A
$order.out  + 30/30 vertices, named, from 82f0a81:
 [1] 9C  5B  12D 11E 10D 7A  11D 8C  6A  9B  9E  7B  6B  6C  8B  12C
[17] 12B 10E 9D  8A  7C  12A 11C 9A  5C  10C 5A  -   11A 11B
$dist
10D   -  5B  9C 12B 12C 11E 12D 11D  6C  6B  8C  7A  6A  7B  8B  9E
  2   0   1   2   3   4   1   2   3   2   3   2   1   1   4   1   5
 9B  9D 10E 12A  9A 11C  7C  8A  5C 11B 11A  5A 10C
  6   1   2   4   2   3   1   2   1   0   0   1   2
```

Din faptul că '–' se găsește – o singură dată, în acest caz – în interiorul vectorului `$order.out` (și nu la sfârșit, ca în cazul lui g35), deducem că g12 are *două* componente conexe; a doua, formează un circuit compus din două vârfuri și extrăgându-le de la sfârșitul lui `$order` (deci, 11A ⟶ 11B ⟶ 11A), rămânem cu problema găsirii circuitelor care conțin '–' (și o putem trata cum am arătat mai sus pentru g35).

Folosim cam cum am arătat mai sus, cei trei vectori returnați de `dfs()`, pentru a constitui o listă `Path` a circuitelor grafului respectiv, din care însă ignorăm nodul '–' – fiindcă în fond, *clasele* ne interesează, nu orele libere; mai jos, notăm prin `G` unul dintre grafurile g35, g12 (sau oricare alt graf al lecțiilor din două coloane):

```
ord <- DFS$order
out <- DFS$order.out
dst <- DFS$dist
dst <- dst[dst == 1]
idx <- which(names(ord) %in% names(dst))
if(! is_connected(G, mode="weak")) {
    idx1 <- which(out %in% V(G)['-'])
    idx <- idx[idx < idx1[1]]
    for(i in seq_along(idx1))
        idx <- c(idx, idx1[i] + i)
}
idx <- c(idx, length(out) + 1)
len <- length(idx)
Path <- list()
for(i in 1:(len-1))
    Path[[i]] <- ord[idx[i] : (idx[i+1]-1)]
```

```
#print(Path)
```

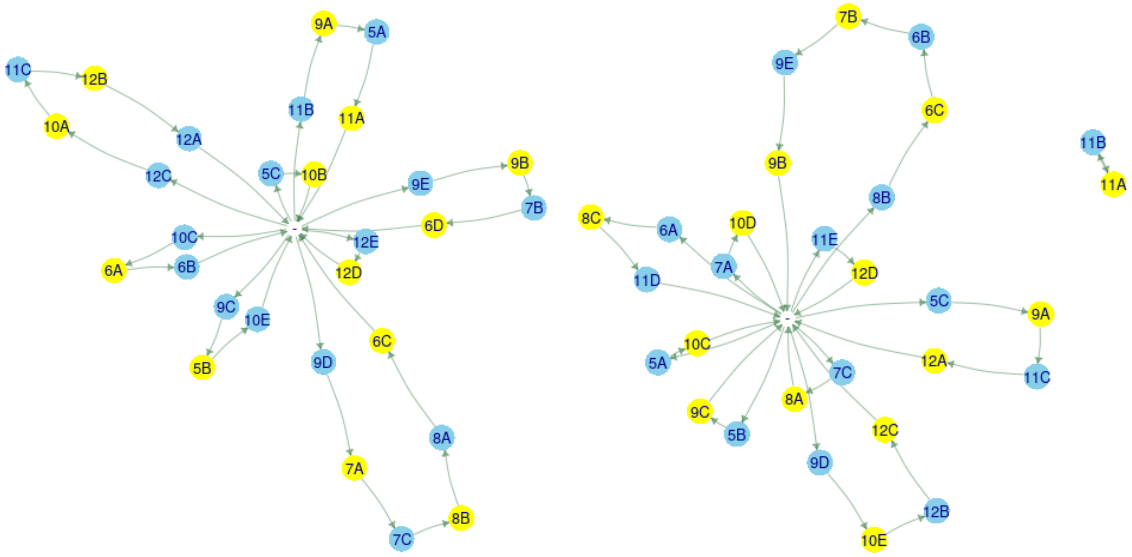
Obs. Puteam proceda mai direct: descompunem întâi în componente conexe, folosind `decompose(G, mode="weak")`; circuitele care trec prin '–' (aflat primul în `$order` și ultimul în `$order.out`, pentru una dintre componentele rezultate) rezultă fără subtilități (spre deosebire de cazul secvenței de mai sus), iar celelalte sunt imediate (toate vârfurile componentei care nu conține '–' formează un circuit al grafului inițial).

Dacă (în loc să afișăm `Path`) plotăm grafurile respective – vom observa – marcând clasele după coloana în care se află – că drumurile obținute în `Path` sunt *lanțurile Kempe* ale lui `G`:

```
for(i in 1:length(Path))  # "colorează" clasele după coloana fiecăreia
    for(j in 1:length(Path[[i]]))  # (pe fiecare circuit în parte)
        V(G)[Path[[i]]][j]$color <- ifelse(j%%2==1,"skyblue","yellow")
V(G)['-']$color <- "white"

coords <- readRDS("coords_12.RDS")  # layout_nicely(G)

plot(G, vertex.frame.color=NA,
        vertex.size = ifelse(V(G)$name == '-', 6, 10),
        vertex.label.family="sans", vertex.label.cex=0.8,
        edge.arrow.size=0.25, edge.arrow.width=2,
        edge.arrow.mode=2, edge.curved = 0.1,
        edge.color = grDevices::adjustcolor(rgb(.25,.5,.3),alpha=0.5),
        layout = coords
)
# saveRDS(coords, file="coords_35.RDS")
```



În stânga este redat graful conex `g35`, iar în dreapta avem graful cu două componente conexe `g12`. Observăm că în oricare circuit din aceste grafuri, culorile nodurilor *alternează*; vârfurile marcate "blue" reprezintă clase din prima coloană, iar cele "yellow" – clase din a doua coloană de lecții.

Precizăm doar că la fiecare nouă executare a programului, `layout_nicely()` (sau alta dintre funcțiile care stabilesc după diverși algoritmi, pozițiile în pagină ale vârfurilor) va furniza *alte* coordonate în parametrul `layout` al funcției `plot()`; putem salva coordonatele rezultate într-o execuție sau alta, pentru a le specifica în `layout`, în cazul vreunei noi execuții (în loc de a mai apela `layout_...()`).

Dacă avem un graf ale cărui vârfuri au fost „colorate" (sau etichetate) astfel încât oricare două noduri adiacente să aibă culori diferite, atunci un lanț maximal al grafului, format numai din noduri colorate cu una dintre două culori `q1` și `q2` date, se numește (`q1-q2`)-*lanț Kempe* (altfel spus, un lanț Kempe[7] este o componentă conexă a subgrafului indus de nodurile care țin de două culori date). Într-un asemenea lanț, culorile alternează (fiindcă vârfurile vecine nu pot avea o aceeași culoare) și se pot interverti (fără a afecta graful), fiindcă lanțul este unul maximal.

Intervertirea celor două culori pe oricare dintre aceste lanțuri, asigură schimbarea claselor respective din coloana `q1` (respectiv, `q2`) în coloana `q2` (respectiv, `q1`); dar acțiunea trebuie făcută pe întregul lanț – altfel, operând astfel numai pe un sub-lanț, ar rezulta și vârfuri adiacente de o aceeași culoare.

Pentru un exemplu, să zicem că vrem să mutăm clasa `10A` din ora a 5-a în ora a 3-a; deci am avea de folosit graful `g35` și pe imaginea lui, redată mai sus, vedem că `10A` aparține următorului (3-5)-lanț Kempe:

```
> K <- c("12C", "10A", "11C", "12B", "12A")
```

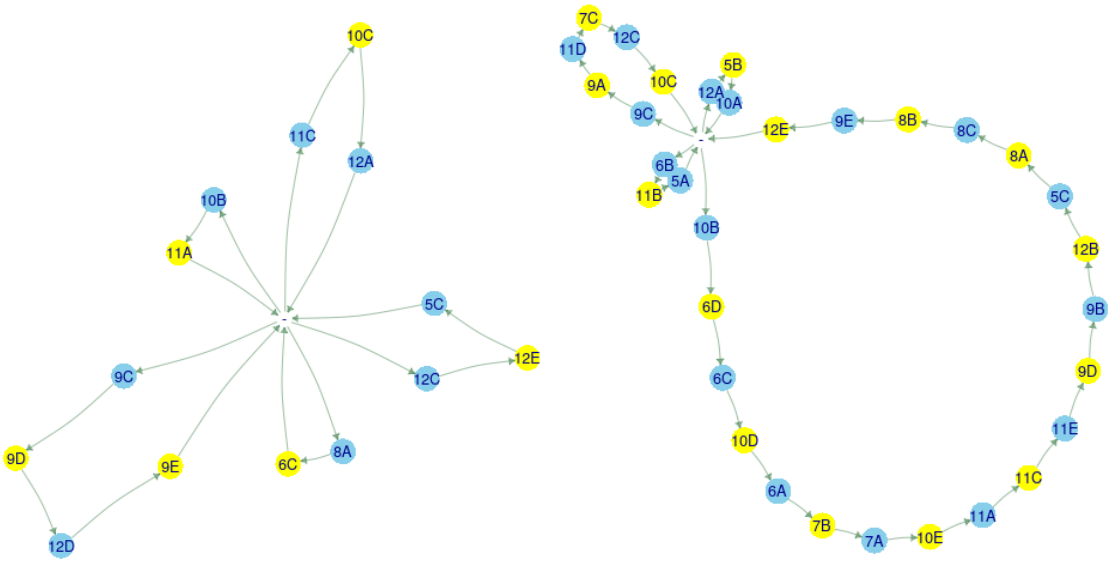
Depistăm în tabelul `EDG` al tuturor lecțiilor din coloanele 3 și 5, acele linii pe care figurează (să zicem, în coloana '`q1`') clasele din lanțul `K` și le afișăm în cele două ordini posibile de coloane:

```
> E1 <- EDG[EDG[, 1] %in% K, ]  # liniile (q1 q2) aferente lanțului K
> E2 <- E1[, 2:1]  # liniile (q2 q1), rezultate mutând 10A din q2 în q1
> print(cbind(E1, E2))  # redă alipit coloanele din E1 și E2
    q1    q2      q2    q1
p22 "12A" "-"     "-"   "12A"
p15 "12B" "12A"   "12A" "12B"
p09 "10A" "11C"   "11C" "10A"
p29 "12C" "10A"   "10A" "12C"
p11 "11C" "12B"   "12B" "11C"
```

[7] https://en.wikipedia.org/wiki/Kempe_chain

Se vede că mutarea clasei respective în cealaltă coloană (fără a crea suprapuneri) este totuna, cu a comuta pe cealaltă culoare vârfurile din lanțul K.

Ar avea importanță, alegerea coloanelor între care să mutăm clase:



De exemplu este mai ușor să mutăm clase din coloana 1 în coloana 4 – conform primului graf redat mai sus, pe care avem numai lanțuri Kempe *scurte* – decât din coloana 4 în coloana 5 (al doilea graf, cu un lanț Kempe foarte lung).

Acestea ar fi aspectele „teoretice” privitoare la mutarea (unei clase mai sus, sau a unui profesor) dintr-o coloană în alta; când va fi mai încolo, să formulăm funcțiile de mutare – va trebui să ținem seama de anumite „asperități”, de exemplu: o clasă care are numai 4 ore, *nu* va putea fi mutată în coloana a 5-a (sau, pentru alt exemplu: dacă lecția aparține unui profesor fictiv, atunci poate fi mutată numai dacă prin aceasta *nu* se va crea o suprapunere ascunsă).

7 Corectarea suprapunerilor ascunse, induse de cuplaje

Ne propunem un program care să corecteze, dar foarte repede, *majoritatea* suprapunerilor ascunse existente; una-două suprapuneri care mai rămân eventual, pe o zi sau alta, pot fi corectate apoi „manual” (mult mai simplu, decât să anticipăm în program, toate excepțiile); vom avea 4 funcții „globale” (și câteva „locale”) și o mică secțiune finală care le angajează efectiv, pe setul de orare zilnice încărcat.