

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ



Lucrare de licență

Cadru de lucru pentru rezolvarea problemelor de tip Stable-Matching

propusă de:

Student: Andreea-Eveline Giosanu

Coordonator științific: Lect. dr. Frăsinaru Cristian

Sesiunea: iulie 2017

Cadru de lucru pentru rezolvarea problemelor de tip Stable-Matching

Student: Andreea-Eveline Giosanu

Coordonator științific: Lect. dr. Frăsinaru Cristian

Sesiunea: iulie 2017

DECLARAȚIE PRIVIND ORIGINALITATEA ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul “Cadru de lucru pentru rezolvarea problemelor tip Stable-Matching” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, 3 iulie 2017

Absolvent *Andreea – Eveline Giosanu*

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul “Cadru de lucru pentru rezolvarea problemelor tip Stable-Matching”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la “Universitatea Alexandru Ioan Cuza” din Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, 3 iulie 2017

Absolvent *Andreea – Eveline Giosanu*

(semnătura în original)

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, *Andreea-Eveline Giosanu*.

Încheierea acestui acord este necesară din următoarele motive:

- proiectul de față prezintă un potențial pentru dezvoltare în viitor, putând fi folosit chiar în cadrul facultății, consider deci că este necesar un astfel de acord;
- toate resursele folosite în realizarea proiectului au fost menționate;
- tot ce este prezentat în lucrarea de față a fost construit de către mine, utilizând materialele necesare, sub îndrumarea domnului profesor coordonator *Cristian Frăsinaru*.

Iași, 3 iulie 2017

Decan *Adrian Iftene*

Absolvent *Andreea – Eveline Giosanu*

(semnătura în original)

(semnătura în original)

Cuprins

Introducere.....	7
1 Stable Matching.....	10
1.1 Caracteristici generale.....	10
1.2 One-to-one	11
1.3 One-to-many	20
1.4 Many-to-many	22
1.4.1 Stable - Allocation	22
1.4.2 Two-sided Market sau Teachers-Repartition	25
2 Modelarea problemelor de Stable-Matching prin reprezentări XML. Instanțierea problemelor	28
2.1 Schema XSD a modelului de bază	28
2.2 Particularitățile problemelor definite prin structura XML.....	30
2.3 Specificații importante	33
3 Arhitectură și implementare	33
3.1 Schema generală a funcționării proiectului	34
3.2 Structura proiectului	35
3.3 Detalii de implementare al algoritmilor.....	38
3.4 Serviciul REST	39
3.5 Tehnologii și platforme utilizate	40
4 Lucrul cu serviciul de <i>Stable-Matching</i>	42
5 Concluzii și perspective de viitor	46
6 Bibliografie	47

Introducere

Motivație

Problemele de *Stable-Matching* acoperă o mare varietate din situațiile din ziua de astăzi. Problema clasică de construire a unei distribuții stabile între entități din clase diferite se întâlnește, în general, sub diferite forme în realitate (spre exemplu: distribuirea studenților la opționale, a rezidenților la spitale, a profesorilor la diferite școli etc). În lucrarea de față se vor prezenta diferite variante ale algoritmilor de *Stable-Matching*, având ca și suport practic implementarea unui cadru de lucru care le tratează.

Problema construirii unei repartiții stabile a apărut în 1962, introdusă de Gale și Shapley, în lucrarea "*College admissions and the stability of marriage*" [1] care a dezvoltat ulterior un interes sporit din partea multor cercetători, fiind aplicabilă în viața reală.

Două seturi de entități, *femei* și *bărbați*, fiecare element având specificată o anumită listă de preferințe cu corespondenți din setul opus, se doresc a fi repartizați într-un *Stable-Matching*. Stabilitatea este dată de inexistența a două elemente care s-ar prefera reciproc față de cei cu care au fost asigurați. Varianta inițială a problemei face parte din clasa *one-to-one* de instanțe și tratează doar cazul unei distribuții între două elemente din clase diferite. Problema s-a extins și la cazuri mai complexe, *one-to-many* și *many-to-many*.

O problema reală a fost sesizată în New York, unde, în fiecare an, elevii care încheiau clasa a 8-a, trebuiau să depună aplicații pentru liceul dorit, completând o listă cu câte 5 preferințe. În primul tur, liceele își alegeau preferații din toți aplicanții iar elevii care nu au fost acceptați la niciun liceu din cele specificate, intrau în următorul tur. Statistica a arătat că majoritatea participanților care nu aveau performanțe deosebite nu au ajuns la vreun liceu menționat în preferințele lor. S-a ajuns, astfel, la construirea unor liste care nu reflectau neapărat aspirațiile reale ale aplicanților ci dorința de a fi siguri că vor ajunge la un liceu

orecum potrivit din punctul lor de vedere. Și, prin urmare, încă din primul tur puteau exista elevi care să ajungă la un liceu mai puțin prestigios deși aveau un profil potrivit pentru unul mai bun, aceștia optând pentru siguranța obținerii unui liceu aproape bun decât pentru a risca să ajungă la un liceu (mai rău) care nu se afla printre preferințele lor. Astfel, s-a încercat găsirea unei modalități noi de distribuții care să reflecte meritele reale ale aplicanților [17].

În lucrarea de față voi prezenta o mare varietate de probleme din această clasă și voi arată cum am pus în practică tot ce am studiat din diferite articole care tratează această temă. Astfel:

- în primul capitol, **Stable – Matching**, am prezentat partea teoretică, atât definiții, tipuri de probleme, exemplificări de instanțe cât și algoritmi care construiesc astfel de distribuții stabile;
- în capitolul al doilea, **Modelarea problemelor de Stable-Matching**, am descris modul în care am construit structura XML-ului care va transpune diferite instanțe de probleme;
- în secțiunea **Arhitectură și implementare** am oferit o imagine de ansamblu a felului în care a fost construit și organizat proiectul;
- în ultima parte este înfățișat modul în care se poate utiliza produsul pus la dispoziție prin intermediul serviciului REST.

Contribuții

Proiectul a pornit de la ideea necesității unui serviciu care să ofere posibilitatea rezolvării variațiilor de problemelor ce au ca și scop construirea unei distribuții între două mulțimi de elemente diferite.

În realizarea acestui proiect a trebuit să încep prin citirea documentației de specialitate pentru a putea pune cap la cap o multitudine de algoritmi ce tratează probleme din aceeași clasă. Documentația a reprezentat elementul fundamental în conceperea unui model de structură al proiectului, bineînțeles, și în construirea reprezentărilor instanțelor sub formă de XML. Modelarea instanțelor sub formă de XML s-a putut realiza și studiind principii esențiale în vederea descoperirii celei mai bune structuri, cum sunt prezentate, spre exemplu în [12].

Pe lângă construirea XML-urilor și implementare algoritmilor, a trebuit să învăț și un framework nou, anume SpringBoot, pentru a putea construi serviciul REST într-un mod cât mai practic pentru a facilita extinderea proiectului în viitor.

1 Stable Matching

Problemele de tip *Stable-Matching* sunt înțelese, în domenii precum: matematică, informatică, științe economice, ca fiind probleme de construire a unei distribuții stabile între două seturi de elemente. *Stable-Marriage* reprezintă varianta de bază a problemelor de acest tip.

Problemele de *Stable Matching* se împart în trei clase:

- *one-to-one* (ex: **Stable-Marriage**)
- *one-to-many* (ex: **Hospitals-Residents** problem, **College-Admission** problem, **Courses-Repartition** problem)
- *many-to-many* (ex: **Stable Allocation**, **Two-sided Market**)

1.1 Caracteristici generale

Notatii:

- **SM** = Stable Matching
- **M** = matching
- $M(e_1)$ = elementul cu care a fost distribuit e_1 , $e_1 \in S_1$ în matching-ul rezultat (având perechea (e_1, l_1) , spunem că $M(e_1) = l_1$)
- $P(e_1, l_1) = p$, p = nivelul de preferință al elementului e_1 pentru l_1 , unde $e_1 \in S_1$ și $l_1 \in S_2$
- $C(e_1) = c$, c = capacitatea maximă a elementului, $e_1 \in S_1$

Definiție SM:

- $\mathbf{M} = \{m_1, m_2, \dots, m_n\}$ un set de n elemente, $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ un set de n elemente
- \mathbf{M} este un *matching* peste cele 2 seturi de elemente (\mathbf{M}, \mathbf{W}) , reprezentând un set de perechi (m, f) , unde $m \in \mathbf{M}$ și $f \in \mathbf{W}$
- \mathbf{M} este stabilă dacă niciuna dintre perechi nu este pereche blocantă, adică trebuie să fie respectată condiția:

$$(m, w) \neq \text{pereche blocantă} \forall m \in \mathbf{M}, w \in \mathbf{W}, (m, w) \in \mathbf{M}$$
- $(m, f) = \text{pereche blocantă}$ dacă au loc următoarele [3]:
 - $M(m) \neq w$;
 - $P(m, w) > P(m, M(m))$;
 - $P(w, m) > P(w, M(w))$.

Sensul optimalității SM-ului

- pentru construirea unui **SM**, algoritmul de bază are ca și principiu adresarea propunerilor în funcție de ordinea elementelor în listele de preferințe
- aceste propuneri se fac într-un singur sens; spre exemplu dacă setul $M(men)$ face propuneri spunem ca **SM** este *men-optimal*
- s-a demonstrat că un **SM** *men-optimal* este cel mai puțin optim dintre toate configurațiile care ar fi plecat de la propuneri venind din setul opus, deci o distribuție *men-optimal* este cel mai puțin favorabilă pentru setul *women* [3]

1.2 One-to-one

Vom trata pentru început cazul de bază, anume problema *Stable-Marriage*:

Enunț:

Fie 2 seturi, femeii (W) și bărbați (M), fiecare având aceeași dimensiune, n . Fiecare element din cele 2 seturi prezintă câte o listă, ordonată strict descrescător, cu preferințele elementelor corespunzătoare setului opus.

Cerință:

Să se construiască un **SM** peste cele două seturi.

Exemplificare:

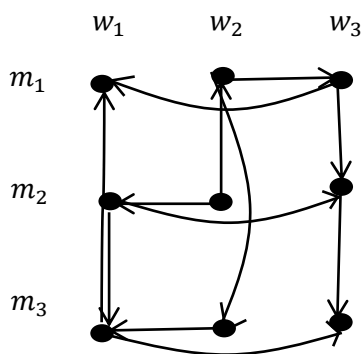
Fie seturile de elemente $M = \{m_1, m_2, m_3\}$, $W = \{w_1, w_2, w_3\}$. Și următoarele liste de preferințe¹:

Setul M :	Setul W :
$P(m_1): w_1, w_3, w_2$	$P(w_1): m_1, m_3, m_2$
$P(m_2): w_3, w_1, w_2$	$P(w_2): m_3, m_1, m_2$
$P(m_3): w_3, w_1, w_2$	$P(w_3): m_3, m_2, m_1$

Se dorește construcția lui \mathbf{M} , unde $\mathbf{M} = \{(m_i, w_j) \mid m_i \in M \wedge w_j \in W, i = \overline{1,3} \wedge j = \overline{1,3}\} \wedge \mathbf{M} = \text{stabilă}$.

¹ Elementele sunt afișate în ordine descrescătoare, ex. $P(b_1, f_1) > P(b_1, f_3)$

Reprezentarea sub formă de graf² a instanței:



În graf sunt desenate arcuri între noduri, reprezenând preferințe astfel:

- Un arc care pleacă din nodul x la un nod y pe linia a se interpretează ca $P(a, x) < P(a, y)$
- Analog și pe coloane, un arc care pleacă din nodul a la un nod b , pe o coloană x , înțelegem $P(x, b) < P(x, a)$
- În caz de nu există un nod la intersecția unei linii cu o coloană, se va înțelege că cel puțin una din cele 2 elemente (date prin id-ul coloanei și id-ul liniei) nu are niciun fel de afinitate pentru cealaltă

Algoritmul GS

Gale și Shapley, în anul 1962, prezentau un algoritm care ar crea un astfel de **SM** [1]; demonstrând, chiar, că orice tip de problemă prezintă cel puțin un **SM**. Complexitatea timp a algoritmului este $O(n^2)$, n fiind dimensiunea fiecărui set în parte.

Descriere:

- Plecând de la una din mulțimile de elemente, se fac, pe rând, „propuneri”
- Pentru elementul curent, din coada elementelor încă libere, se iau partenerii preferați în ordine descrescătoare
- Dacă primul element din lista de preferințe este liber, atunci se adaugă în match-ul curent perechea (m, w)

² Reprezentare construită conform indicațiilor prezentate în [2]

- În caz contrar, adică elementul w nu este disponibil, fiind deja repartizat cu altcineva, se va verifica dacă nu cumva w , l-ar prefera pe actualul pretendent față de cel cu care este deja repartizat, adică $P(w, m) > P(w, m_1)$
- Dacă are loc $P(w, m) > P(w, m_1)$ atunci f va fi distribuit cu b ($M = M \cup \{(w, m)\}$), iar m_1 va fi adăugat din nou în coada de elemente libere ($L = L \cup m_1$), urmând să facă din nou propuneri
- Dacă **nu** are loc $P(w, m) > P(w, m_1)$, atunci se va reîntra în bucla *WHILE*(*m's preferences list* $\neq \emptyset$) și, în caz de m încă mai are elemente în lista de preferințe, acesta îi va face o propunere noului element, urmând din nou pașii descriși mai sus
- Când m nu mai are preferințe înseamnă că a fost refuzat de toate elementele pe care le-ar fi acceptat, deci nu va mai avea cum să aibă vreo pereche, prin urmare nu îl mai adăugăm în coada de elemente libere
- Procesul acesta se va executa pentru toate elementele din L

Algoritmul GS

```

M = ∅
F = men
set all L's elements to be available
WHILE L ≠ ∅
    m = F.getCurrentElement()
    WHILE m's preference list ≠ ∅ AND m = available
        w = m.preferences.poll()
        IF w = available
            M = M ∪ {(m, w)}
            set m's availability to False
        ELSE
            m1 = current match for w
            IF P(w, m) < P(w, m1)
                CONTINUE
            M = M \ {(m1, w)}
            F = F ∪ m1
            M = M ∪ {(m, w)}
            m set availability to False
            m1 set availability to True
return M

```

În pseudocod avem:

- F = lista elementelor disponibile; va fi inițializată cu mulțimea elementelor din M (men)
- \mathbf{M} = matching-ul ce va fi rezultat, inițial fiind mulțimea vidă (\emptyset)

Construcția lui \mathbf{M} cu algoritmul GS pentru instanța prezentată:

Pasul 1: inițializarea

- $\mathbf{M} = \emptyset$
- $F = \{m_1, m_2, m_3\}$
- m_1, m_2, m_3 = disponibile
- w_1, w_2, w_3 = disponibile

Pasul 2: propunerile

I. m_1 face propuneri:

- $F = F \setminus m_1 = \{m_2, m_3\}$
- m_1 propune ca și partener pe w_1
- w_1 este disponibil și îl acceptă pe m_1 ca și partener
- $\mathbf{M} = \{(m_1, w_1)\}$

II. m_2 face propuneri:

- $F = F \setminus m_2 = \{m_3\}$
- m_2 propune ca și partener pe w_3
- w_3 este disponibil și îl acceptă pe m_2 ca și partener
- $\mathbf{M} = \mathbf{M} \cup \{(m_2, w_3)\} = \{(m_1, w_1), (m_2, w_3)\}$

III. m_3 face propuneri:

- $F = F \setminus m_3 = \emptyset$
- m_3 propune ca și partener pe w_3
- w_3 nu mai este disponibil și avem $P(w_3, m_2) < P(w_3, m_3)$, deci w_3 îl va accepta pe m_3 și îl va respinge pe m_2
- $\mathbf{M} = \mathbf{M} \setminus \{(m_2, w_3)\} = \{(m_1, w_1)\}$

- $\mathbf{M} = \mathbf{M} \cup \{(m_3, w_3)\} = \{(m_1, w_1), (m_3, w_3)\}$
- $F = F \cup m_2 = \{w_2\}$

IV. m_2 face propuneri după ce a fost respins de f_3 :

- $F = F \setminus m_2 = \emptyset$
- m_2 propune ca și partener pe w_1
- w_1 nu mai este disponibil și avem $P(w_1, w_1) > P(w_1, m_2)$, deci w_1 nu îl va accepta pe m_2

V. m_2 face propuneri după ce a fost respins de w_1 :

- m_2 propune ca și partener pe w_2
- m_2 este disponibil și îl acceptă pe w_2 ca și partener
- $\mathbf{M} = \mathbf{M} \cup \{(m_2, w_2)\} = \{(m_1, w_1), (m_3, w_3), (m_2, w_2)\}$

Pasul 3: întoarcerea match-ului

$\mathbf{M} = \{(m_1, w_1), (m_3, w_3), (m_2, w_2)\}$

Extensii ale problemei Stable-Marriage

Problema de bază *Stable-Marriage* prezintă două seturi de elemente, ambele cu câte n componente. Fiecare element prezintă câte o listă strict ordonată cu cele n elemente din setul opus. Bineînțeles, în practică nu pot fi îndeplinite oricând toate aceste condiții, de aceea există și extensii în care condițiile sunt mai permissive.

I. *Stable-Marriage* cu liste incomplete (SMI):

- elementele nu mai sunt restricționate la construirea de liste cu toate elementele din setul opus ($size(P(w)) \leq n$)

II. *Stable-Marriage* cu indiferențe (SMT):

- nu mai există restricția ca listele de preferințe să fie ordonate crescător, în listele de preferințe pot apărea elemente la același nivel, adică avem condiții de forma $P(m, w_i) \geq P(m, w_j) (\forall i, j = \overline{1, n} \wedge i \neq j)$

III. *Stable-Marriage* cu liste incomplete și indiferențe (SMTI):

- au loc condițiile prezentate la I și II

Pentru aceste tipuri de instanțe *stabilitatea* matching-ului ce va fi construit este ușor diferită de varianta clasică. Se definesc, deci, trei tipuri de stabilitate [3]:

- *stabilitatea slabă*³: definită ca în cazul instanțelor de *Stable-Marriage* simple
- *stabilitatea puternică*⁴: \mathbf{M} = puternic stabilă dacă $\nexists (m, w)$ pereche blocantă, $(m, w) \in \mathbf{M} (\forall m \in M \wedge w \in W)$, în acest caz $(m, w) = \text{pereche blocantă}$, dacă au loc următoarele:
 - $\mathbf{M}(m) \neq w$;
 - $P(m, w) > P(m, \mathbf{M}(m))$;
 - $P(w, m) \geq P(w, \mathbf{M}(w))$.
- *super stabilitatea*⁵: \mathbf{M} = super stabilă dacă $\nexists (m, w)$ pereche blocantă, unde $(m, w) \in \mathbf{M} (\forall m \in M \wedge w \in W)$, în acest caz $(m, w) = \text{pereche blocantă}$, dacă au loc următoarele:
 - $\mathbf{M}(m) \neq w$;
 - $P(m, w) \geq P(m, \mathbf{M}(m))$;
 - $P(w, m) \geq P(w, \mathbf{M}(w))$.

De acum vom folosi notația *SMTI* pentru a îngloba toate tipurile de extensii de *Stable-Matching*, anume *SMT*, *SMI* și *SMTI*. De asemenea, restul tipurilor de probleme au fost tratate fără a fi respectate restricțiile prezentate de problema clasică de *Stable-Matching*.

Problemele *SMTI* sunt instanțe de *Stable-Marriage* în care există conceptul de acceptabilitate între entități. Cu alte cuvinte, dacă un element m nu prezintă în lista lui de preferințe un element f spunem că m nu acceptă f , prin urmare, în niciun \mathbf{SM} ce va fi creat nu va putea exista o pereche formată din cele două entități.

³ *weak-stability* în eng.

⁴ *strong-stability* în eng.

⁵ *super-stability* în eng.

Optimizări ale problemelor de tip SMTI

În cazul problemei clasice, Gale și Shapley au propus un algoritm care construiește un **SM** ca având dimensiunea maximă posibilă, anume n [2]. Când se tratează cazuri ce nu îndeplinesc condițiile, fiind vorba de instanțe de *SMTI*, problema construirii unui **SM** maximal devine mai complicată.

Au fost propuși algoritmi care construiesc aproximări ale unui **SM** maximal, spre exemplu Irving și Manlove propun un algoritm cu factor de aproximare de $5/3$ [4], dar cu condiția ca doar unul din cele două seturi să conțină indiferențe în listele de preferințe.

Zoltán Király oferă o abordare în timp liniar care are factor de aproximare de $3/2$ [5] și chiar fără condiția impusă de algoritmul precedent. De asemenea, există multe alte propuneri însă cu un factor de aproximare mai mic decât cele menționate.

Având în vedere că cel din urmă prezintă o aproximare mai bună, am decis să implementez această variantă optimizată a instanțelor de tip *SMTI* și ca urmare îl voi prezenta în continuare.

Algoritmul lui Király pentru probleme de tip SMTI

Înainte de a prezenta ideea propusă de Király, vom defini *factorul de aproximare*. *Factorul de aproximare* al unui algoritm de optimizare, pentru instanțe de *SMTI*, desemnat prin f , este acel număr pentru care algoritmul returnează un matching M cu dimensiunea $\text{size}(M) \geq 1/f$. [5]

Király a pornit de la faptul că multe elemente din primul set (cel care face propuneri) rămân fără parteneri la final, unii dintre ei fiind preferați în mod egal cu cei cu care au fost distribuiți pretenții lor, adică $P(f, m) = P(f, M(f))$. Dezavantajul pornește de la faptul că dacă un pretendent curent este repartizat cu un element pe care îl preferă la fel ca și pe cel care face noua propunere, acesta îl refuză pe cel din urmă.

Principiul primul venit, primul servit. Asta reprezintă o problemă deoarece pot exista cazuri când cel care a fost refuzat să nu mai poată fi acceptat de altcineva, în timp ce, cel pentru care a fost refuzat, să mai aibă și alte șanse. Astfel, dimensiunea matching-ului are de suferit.

Algoritmul propus pornește de la un matching deja construit cu algoritmul GS pentru o instanță de *SMTI* și, având toate elementele care nu au fost distribuite, se *repromovează*. Conceptul de bază este, de fapt, “distrugerea indiferențelor”⁶ sau “ruperea egalităților”.

Acest lucru se realizează prin creșterea preferinței cu un ε , unde $\varepsilon = 1/2$, pentru toate elementele încă disponibile, doar unde se găsesc egalități. Adică dacă $P(w, m_1) = P(w, m_2)$, iar m_2 este disponibil, $P(w, m_2) = P(w, m_2) + \varepsilon$. Deci când m_2 va face din nou propuneri w îl va prefera pe m_2 față de m_1 . Valoarea lui ε este rațională și subunitară deoarece se dorește “spargerea egalităților” nu creșterea cu un nivel al elementului *promovat*.

O dată ce un element a fost deja promovat el nu va mai putea fi promovat încă o dată, chiar dacă în matching-ul nou este tot nerepartizat.

Algoritmul se oprește când orice element care este liber, la un moment dat, a fost promovat.

Algoritmul Kiraly

```

V = {v[m] = False | m ∈ men}
WHILE not all available men have been promoted
    M = run GS
    FOR all available m elements
        IF v[m] = False
            set v[m] to True
            raise the level of m with  $\varepsilon$  for all the ties

```

În pseudocod avem definite:

- V = mulțimea care reține care element din M (*men*) a fost vizitat deja
- M = matching-ul rezultat rulând algoritmul GS
- după creșterea nivelului lui m în preferințele celor din setul W , se va construi un nou matching cu listele de preferințe actualizate

⁶ *break the ties* în eng.

- se vor re-promova de data aceasta elementele care nu au fost până acum promovate și care nu au partener în noul matching
- procesul se va repeta până nu vor mai exista elemente disponibile nepromovate

GS și Kiraly. Comparație

Avem instanța: $W = \{w_1, w_2, w_3, w_4, w_5\}$ $M = \{m_1, m_2, m_3, m_4, m_5\}$

Setul W :

$P(w_1): m_1, m_2, m_3, m_4$

$P(w_2): m_2, m_1, m_4, m_3$

$P(w_3): m_2, m_1, m_3, m_4$

$P(w_4): m_2, m_1, m_3, m_4$

$P(w_5): m_3, m_1, m_2, m_4$

Setul M :

$P(m_1): w_1, w_2$

$P(m_2): w_1, (w_3, w_2), w_4$

$P(m_3): w_1, w_2, w_5$

$P(m_4): w_1, w_2$

$P(m_5): w_5, w_2$

Rulând cei doi algoritmi pe această instanță avem următoarele output-uri:

```

---GS SOLVER---

Match:
w1 + m1
w2 + m2
w5 + m3
Free elements: w3 w4

Process finished with exit code 0

```

```

---KIRALY SOLVER---

Match:
w1 + m1
w3 + m2
w5 + m3
w2 + m4
Free elements: w4

Process finished with exit code 0

```

$$|M_{GS}| < |M_K|^7$$

Se observă, deci, că algoritmul lui Király oferă un matching cu o dimensiune mai mare decât algoritmul obișnuit. Pentru a crea matching-uri cu un factor de aproximare mai bun, algoritmul lui Király a fost folosit și pentru instanțele de *one-to-many*.

⁷ Prin $|M|$ = cardinalul mulțimii M

1.3 One-to-many

În această secțiune vom trata problema *Hospitals-Residents*.

Enunț:

Fie două seturi: spitale (H) și rezidenți (R). Fiecare element din cele două seturi prezintă câte o listă, ordonată descrescător, cu preferințele elementelor corespunzătoare setului opus. Fiecare spital prezintă o capacitate, reprezentând numărul de posturi disponibile.

Cerință:

Să se construiască un **SM** peste cele două seturi.

Particularități

- un set de elemente prezintă o caracteristică, *capacitatea*, care desemnează numărul maxim de locuri disponibile/numărul maxim de perechi posibile pe care-l poate forma
- se introduce noțiunea de *cupluri* prin care se înțelege că două elemente (din setul celor care nu au capacitatea specificată) prezintă aceleași preferințe și doresc să fie repartizate în același loc

Exemplificare

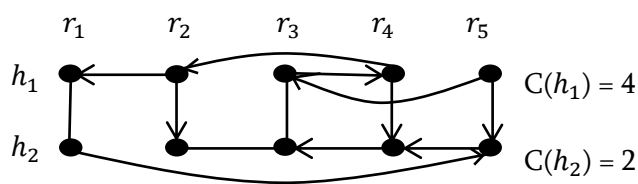
Fie seturile de elemente $H = \{h_1, h_2\}$, $R = \{r_1, r_2, r_3, r_4, r_5\}$ și următoarele liste de preferințe:

Setul H :	Setul R :	Cupluri:
$P(h_1): r_1, r_2, r_4, r_3, r_5$	$P(r_1): (h_1, h_2)$	(r_1, r_3)
$P(h_2): (r_3, r_2), r_4, r_5, r_1$	$P(r_2): h_2, h_1$	
$C(h_1) = 4$	$P(r_3): (h_1, h_2)$	
$C(h_2) = 2$	$P(r_4): h_2, h_1$	
	$P(r_5): h_2, h_1$	

Se dorește construcția lui \mathbf{M} , unde $\mathbf{M} = \{(r_i, h_j) \mid r_i \in R \wedge h_j \in H, i = \overline{1,3} \wedge j = \overline{1,3}\} \wedge \mathbf{M} = \text{stabilă}$.

Introducerea noului element care trebuie luat în calcul, *capacitatea*, nu reprezintă o schimbare drastică în comportamentul algoritmului de bază. De aceea, elementele cu capacitate sunt luate în calcul ca fiind $capacitate(h)$ elemente în locul unui singur element h .

Reprezentarea instanței sub formă de graf conform [2]:



- o muchie orizontală/verticală între 2 noduri reprezintă 2 elemente preferate în mod egal de elementul desemnat de linie/coloană

Algoritm

Algoritmul se construiește pornind de la algoritmul GS, rezidenții fiind cei care fac propuneri. Spitalele acceptă în numărul locurilor disponibile și pe baza preferințelor. În cazul cuplurilor dacă un spital ar accepta doar unul din cei doi, cuplul va trece la următoarea opțiune. Cuplul este tratat ca un singur element dar care ocupă 2 locuri în loc de unul. [6]

Rezultatul algoritmului pentru instanța propusă:

```

---HR Solver---
Match:
r1 + h1
r3 + h1
r2 + h2
r4 + h2
r5 + h1
Free elements: none.

Process finished with exit code 0

```

$$\mathbf{M} = \{(r_1, h_1), (r_3, h_1), (r_2, h_2), (r_4, h_2), (r_5, h_1)\}$$

Problemele **College-Admission**, **Courses-Repartition** sunt tratate exact în același mod, în fond toate reprezentând instanțe de *one-to-many*.

1.4 Many-to-many

În această secțiune vorbim despre două probleme, fiecare cu propriile particularități, **Stable-Allocation** și **Two-Sided Market/Teachers Repartition**.

1.4.1 Stable - Allocation

Enunț:

Fie două seturi: firme (F) și lucrători (W). Fiecare element din cele două seturi prezintă câte o listă, ordonată descrescător, cu preferințele elementelor corespunzătoare setului opus. De asemenea, fiecare element prezintă un număr de unități. Pe lângă unitățile fiecăruia există un număr maxim de unități pe care îl poate oferi un anumit lucrător la o anumită firmă.

Cerință:

Să se construiască un **SM** peste cele două seturi.

Particularități

- numărul de unități semnifică pentru o firmă numărul de ore pe care ar dori să le acopere viitorii angajați ($U(f) > 0, \forall f \in F$)
- pentru lucrători, unitățile semnifică numărul total de ore pe care sunt dispuși să îl ofere ($U(w) > 0, \forall w \in W$)
- numărul maxim de unități dintre o firmă și un lucrător se referă la numărul maxim de ore pe care poate lucrătorul respectiv să le aloce la acea firmă, adică ($U(w, f) \geq 0, \forall w \in W \wedge f \in F$)

Exemplificare

Fie seturile de elemente $F = \{f_1, f_2, f_3\}$, $W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$ și următoarele liste de preferințe:

Setul F :		Setul L :	
$P(f_1): w_3, w_7, w_2, w_1, w_5, (w_4, w_6)$	$U(f_1) = 70$	$P(w_1): f_2, f_3, f_1$	$U(w_1) = 20$
$P(f_2): w_2, w_1, w_5, w_4, w_3, w_6$	$U(f_2) = 55$	$P(w_2): f_2, f_1, f_3$	$U(w_2) = 16$
$P(f_3): w_2, w_7, w_4, w_5, w_3, w_6, w_1$	$U(f_3) = 31$	$P(w_3): f_1, f_2$	$U(w_3) = 15$
		$P(w_4): f_2, f_3, f_1$	$U(w_4) = 14$

Max-unități:	$P(w_5): f_1, f_3, f_2$	$U(w_5) = 19$
$U(f_1, w_i) = 15 \ \forall \ w_i \in L$	$P(w_6): f_3, f_2, f_1$	$U(w_6) = 21$
$U(f_2, w_i) = 12 \ \forall \ w_i \in L$	$P(w_7): f_1, f_2, f_3$	$U(w_7) = 23$
$U(f_3, w_i) = 10 \ \forall \ w_i \in L$		

Se dorește construcția lui \mathbf{M} , unde $\mathbf{M} = \{(w_i, f_j) \mid w_i \in \mathbf{W} \wedge f_j \in \mathbf{F}, i = \overline{1,3} \wedge j = \overline{1,3}\}$
 $\wedge \mathbf{M} = \text{stabilă}$.

Reprezentarea sub formă de graf a acestei instanțe este mai greu de interpretat, nu aduce niciun beneficiu vizual, prin urmare, nu o mai construim.

Algoritm

Băiou și Balinski descriu un algoritm pentru a rezolva această problemă care pleacă de la algoritmul GS [2]. În acest context apare conceptul de alocare de unități între entități din seturi diferite.

Algoritmul *worker-optimal* pleacă de la lucrători care fac propuneri în funcție de preferințe. Orele ce urmează a fi alocate între un anumit lucrător w și o firmă f se alege obținând minimumul dintre $U(w)$, $U(f)$ și $U(w, f)$. Dacă firma f este complet indisponibilă, dar un lucrător nou w' care face o propunere și se află în lista preferințelor mai sus decât un lucrător w căruia deja i-au fost alocate un număr de ore, w va fi respins și va fi readăugat în lista elementelor încă disponibile.

În continuare vom nota:

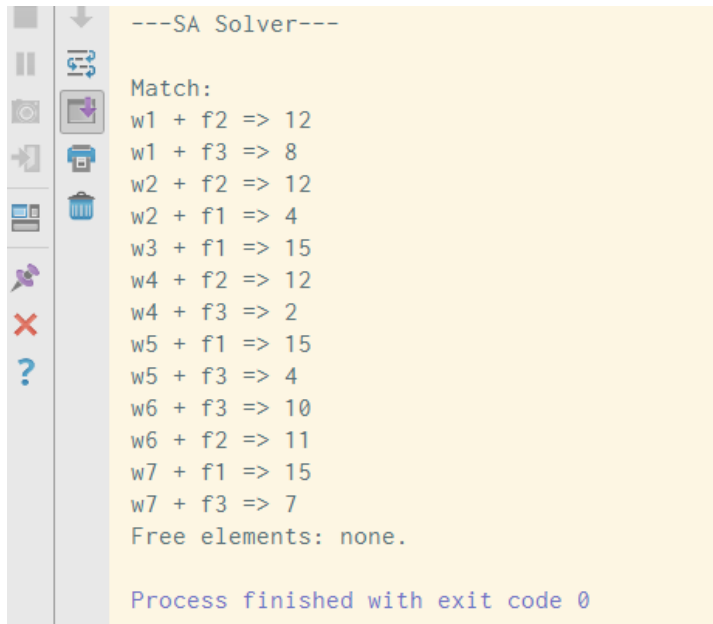
- W = mulțimea lucrătorilor
- F = mulțimea elementelor disponibile; va fi inițializată cu W
- $A(w, f)$ = numărul de unități alocate între w și f

Algoritmul SA

```

F = W
FOR w in F
  WHILE w is available AND w still has preferences
    f = w.preferences.poll()
    IF f does NOT accept w
      CONTINUE
    IF f = available
      u = get min (units left for w, units left for f, (maxunits between w
        and f))
      allocate u units between (w, f)
    ELSE
      w1 = least preferred for f
      IF P(f, w) > P(f, w1)
        eliminate l1 from f
        F = F ∪ w1
        allocate min(units left for w, units got from eliminating w1 from
          f, maxunits between w and f)
      IF hours left for w = 0
        set w to be full
  
```

Rezultatul obținut pentru instanța prezentată:



```

---SA Solver---

Match:
w1 + f2 => 12
w1 + f3 => 8
w2 + f2 => 12
w2 + f1 => 4
w3 + f1 => 15
w4 + f2 => 12
w4 + f3 => 2
w5 + f1 => 15
w5 + f3 => 4
w6 + f3 => 10
w6 + f2 => 11
w7 + f1 => 15
w7 + f3 => 7
Free elements: none.

Process finished with exit code 0
  
```

$$\mathbf{M} = \begin{cases} w_1, f_2 & A(w_1, f_2) = 12 \\ w_1, f_3 & A(w_1, f_3) = 8 \\ w_2, f_2 & A(w_2, f_2) = 12 \\ w_2, f_1 & A(w_2, f_1) = 4 \\ w_3, f_1 & A(w_3, f_1) = 15 \\ w_4, f_2 & A(w_4, f_2) = 12 \\ w_4, f_3 & A(w_4, f_3) = 2 \\ w_5, f_1 & A(w_5, f_1) = 15 \\ w_5, f_3 & A(w_5, f_3) = 4 \\ w_6, f_3 & A(w_6, f_3) = 10 \\ w_6, f_2 & A(w_6, f_2) = 11 \\ w_7, f_1 & A(w_7, f_1) = 15 \\ w_7, f_3 & A(w_7, f_3) = 7 \end{cases}$$

1.4.2 Two-sided Market sau Teachers-Repartition

Enunț:

Fie două seturi: firme (F) și lucrători (W). Fiecare element din cele două seturi prezintă câte o listă, ordonată descrescător, cu preferințe. Listele conțin și *grupuri* de elemente. De asemenea fiecare element prezintă o anumită *capacitate*.

Cerință:

Să se construiască un **SM** peste cele două seturi.

Particularități

- față de **Stable-Allocation**, aici nu mai avem unități, avem ca la **Hospitals-Residents**, capacități; de data aceasta la ambele seturi
- prin *capacitate* se înțelege numărul total de locuri la care ar putea fi angajat un lucrător w , iar pentru o firmă f se înțelege numărul de posturi libere
- *grupul* este o particularitate aparte a acestei probleme, prin ideea de grup nu se creează o legătură atât de strânsă ca la cupluri
- *grupurile* sunt întâlnite doar la nivel de preferințe, adică o firmă f poate menționa că ar prefera un grup format din w_1 și w_2 față de un grup format din w_1 și w_3

Exemplificare

Fie seturile de elemente $F = \{f_1, f_2, f_3, f_4\}$, $W = \{w_1, w_2, w_3, w_4\}$. Și următoarele liste de preferințe⁸:

Setul F :

$P(f_1): < w_1, w_2 >, (< w_1, w_3 >, < w_3, w_4 >), < w_2, w_4 >, < w_1, w_4 >, < w_2, w_3 >$
 $, w_1, w_2, w_3, w_4$

$P(f_2): (< w_1, w_2 >, < w_1, w_4 >), (< w_2, w_3 >, < w_2, w_4 >), < w_3, w_4 >, < w_1, w_3 >$

$P(f_3): < w_1, w_4 >, < w_1, w_2 >, < w_2, w_4 >, w_1, w_2, w_4$

$P(f_4): < w_1, w_2 >, w_1, w_2$

$C(f_1) = 2 \quad C(f_2) = 2$

$C(f_3) = 2 \quad C(f_4) = 2$

⁸ Prin notația $<w, w'>$ se înțelege grupul format din w, w' cu $w, w' \in W$

Setul \mathcal{W} :

$P(w_1): \langle f_3, f_4 \rangle, (\langle f_2, f_3 \rangle, \langle f_2, w_4 \rangle), \langle f_1, f_3 \rangle, \langle f_1, f_2 \rangle, f_1, f_2, f_3, f_4$

$P(w_2): (\langle f_3, f_4 \rangle, \langle f_2, f_4 \rangle), \langle f_2, f_3 \rangle, \langle f_1, f_4 \rangle, (\langle f_1, f_3 \rangle, \langle f_1, f_2 \rangle), f_1, f_2, f_3, f_4$

$P(w_3): (\langle f_1, f_2 \rangle, \langle f_2, f_4 \rangle, \langle f_1, f_3 \rangle), \langle f_2, f_3 \rangle, \langle f_1, f_4 \rangle, \langle f_3, f_4 \rangle, f_1, f_2, f_3, f_4$

$P(w_4): (\langle f_1, f_2 \rangle, \langle f_2, f_4 \rangle, \langle f_3, f_4 \rangle), \langle f_1, f_3 \rangle, \langle f_1, f_4 \rangle, \langle f_2, f_3 \rangle, f_1, f_2, f_3, f_4$

$C(w_1) = 2$

$C(w_2) = 2$

$C(w_3) = 2$

$C(w_4) = 2$

Se dorește construcția lui \mathbf{M} , unde $\mathbf{M} = \{(f_i, w_j) \mid f_i \in \mathbf{F} \wedge w_j \in \mathbf{W}, i = \overline{1,3} \wedge j = \overline{1,3}\} \wedge \mathbf{M} = \text{stabilă}$.

Algoritm

În rezolvarea acestui tip de problemă, algoritmul pornește, de asemenea, de la ideea de „acceptare întârziată”⁹ propusă de Gale și Shapley. Ce are nou acest algoritm este conceptul pe baza căruia se construiește matching-ul, anume prin „preferințe substituibile”¹⁰. În momentul în care o firmă este respinsă de un anumit lucrător, atunci aceasta va elimina, din lista de preferințe, toate grupurile care conțin elementul respectiv.

La la fiecare pas, o firmă face propuneri până nu mai are posturi disponibile. La sfârșitul unei runde de propuneri, angajații, în caz de încă nu au primit suficiente oferte, rămân încă disponibili, iar restul, care au primit prea multe cereri, aleg dintre ofertele primite ce preferă mai mult, refuzând firmele mai puțin dorite. În momentul în care o firmă este respinsă are loc fenomenul de „eliminare” a tuturor grupurilor ce conțin enitatea care a refuzat-o.

Algoritmul se termină când toate firmele și-au ocupat toate posturile și, când cele care încă au locuri disponibile au fost refuzate de toți angajații pe care și i-ar mai fi dorit.

⁹ Deferred Acceptance în eng.

¹⁰ Substitutable Preferences în eng.

În continuare vom defini următoarele:

- W = mulțimea lucrătorilor
- F = mulțimea elementelor disponibile; va fi inițializată cu W
- $c(x)$ = capacitatea unui element x
- prin *group* se înțelege orice tip de element, atât grup format din n entități cât și dintr-un singur element

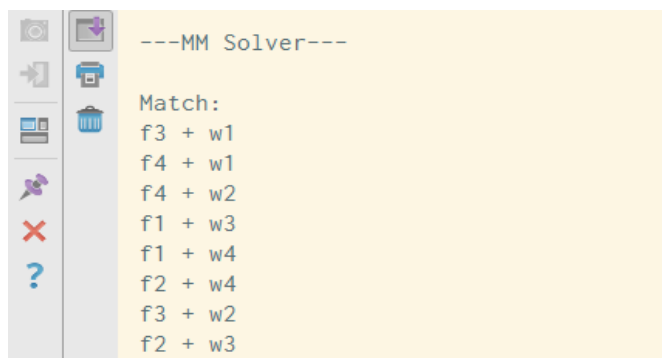
Algoritmul MM

```

DO
  FOR  $f$  in available firms
    WHILE  $f$  is still available AND still has elements to propose to
       $group = f.preferences.poll()$ 
      WHILE not every element in group accepts  $f$ 
         $group = f.preferences.poll()$ 
      assign group with  $f$  in  $M$ 
       $c(f) = s(f) - (\text{size}(group) - \text{size}(\text{elements that were already matched with } f))$ 
      IF  $c(f) = 0$ 
        set  $f$  to be full
  FOR worker  $w$  in current matched workers
    IF  $w$  is oversubscribed
       $w$  rejects all his least preferred firms until is full
      FOR every  $f$  that was rejected by  $w$ 
        eliminate all elements that contain  $w$  in  $f$ 's preferences
         $c(f) = c(f) + 1$ 
        add  $f$  to the available set
  WHILE there are still available firms

```

Rezultatul obținut pentru instanța prezentată:



$$M = \begin{cases} f_1, w_3 \\ f_1, w_4 \\ f_2, w_3 \\ f_2, w_4 \\ f_3, w_1 \\ f_3, w_2 \\ f_4, w_1 \\ f_4, w_2 \end{cases}$$

2 Modelarea problemelor de Stable-Matching prin reprezentări XML. Instanțierea problemelor

Pentru a putea crea un mod cât mai ușor de reprezentare a datelor, instanțierea problemelor se va face pe baza unui XML, acesta dovedindu-se cel mai potrivit. Structura acestuia a fost concepută respectând principiile și regulile stabilite în [a].

Bineînțeles, fiecare tip de problemă prezintă particularitățile sale dar, toate pleacă de la aceeași configurație de bază. În continuare voi prezenta structura generală a XML-ului și particularitățile fiecărei probleme, iar la *Anexe A-D* sunt atașate modele de XML pentru fiecare tip de problemă în parte.

2.1 Schema XSD a modelului de bază

Elementele principale pe care **trebuie** să le conțină XML-ul sunt:

- Tipul problemei (ex: **SM**, **HR**, **MM**, **SA**)
- Definirea seturilor (nume set, dimensiune și elementele propriu-zise)
- Definirea preferințelor:
 - Numele elementului pentru care se definește lista de preferințe
 - Numele elementului curent, nivelul de preferință (1 = cel mai preferat)
 - În caz de avem egalitate, vom menționa acest lucru și vom trece numele tuturor elementelor

De asemenea sunt și elemente care **pot** fi menționate, precum:

- Descriere a problemei
- Dimensiunea (2D/3D)¹¹

Un alt aspect important ce trebuie respectat este faptul că seturile de elemente trebuie definite înainte să fie specificată vreo preferință. Elementele **trebuie** să existe pentru a putea fi sau pentru a putea avea preferințe.

În continuare este prezentată schema XSD a XML-ului de bază, pentru probleme *one-to-one*.¹²

¹¹ Deocamdată biblioteca rezolvă doar probleme 2D, dar există și opțiunea 3D pentru extensii viitoare

¹² Schema a fost generată folosind [13]

```

<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Studio 2017 - Developer Bundle Edition (Trial) 15.1.4.7515
(https://www.liquid-technologies.com) -->
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="problem">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element minOccurs="2" maxOccurs="unbounded" name="set">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string" use="required" />
                <xs:attribute name="size" type="xs:string" use="required" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="preferences">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="element">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="name" type="xs:string" use="required" />
                      <xs:attribute name="level" type="xs:unsignedByte" use="req"/>
                      <xs:attribute name="tie" type="xs:boolean" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="dimension" type="xs:string" use="optional" />
      <xs:attribute name="description" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Schema XSD a modelului de bază

Un exemplu minimal de instanță ar putea fi:

```
<?xml version="1.0" encoding="utf-8"?>
<problem name="SM" dimension="2D" description="Sample of SM instance">
  <set name="women" size="3"> w1, w2, w3 </set>
  <set name="men" size="3"> m1, m2, m3 </set>
  <preferences name="m1">
    <element name="w1" level="1" />
    <element level="2" tie="true">
      w2, w3
    </element>
  </preferences>
  <preferences name="w1">
    <element name="m1" level="1" />
    <element name="m2" level="2" />
    <element name="m3" level="2" />
  </preferences>
</problem>
```

Restricții: *dimensiunea* și *nivelul* trebuie să aibă o valoare pozitivă altfel instanța se va considera invalidă.

2.2 Particularitățile problemelor definite prin structura XML

Definirea capacității

Pentru instanțele de tip *one-to-many* și *many-to-many*, unde este necesar să fie specificate capacitățile elementelor, se prezintă o structură ușor diferită al tagului „<set>”. Fiecare element din set va fi caracterizat prin intermediul tagului „<element>” care va avea ca și atribut „name” și „capacity”.

Definim mai jos $H = \{h_1, h_2\}$, $C(h_1) = 4$ și $C(h_2) = 2$:

```
<set name="hospitals" size="2">
  <element name="h1" capacity="4"/>
  <element name="h2" capacity="2"/>
</set>
```

Restricții: *capacitatea* trebuie să aibă o valoare pozitivă, altfel instanța se va considera invalidă.

Definirea cuplurilor

Cuplurile apar în cazul instanțelor de *one-to-many*. Ele se definesc în interiorul tagului „<group>”, după ce au fost deja definite seturile de elemente și înainte de a fi specificate preferințele.

Definim mai jos cuplurile r_1, r_2 și r_3, r_4 :

```
<group>
  <couple> r1, r2 </couple>
  <couple> r3, r4 </couple>
</group>
```

Restricții: elementele definite sub tagul „<couple>” vor fi tratate ca având aceleași preferințe deci, dacă apar liste pentru ambele elemente, instanța se va considera invalidă. Tagul „<couple>” trebuie să conțină maxim două elemente.

Definirea grupurilor

În cadrul instanțelor de *many-to-many* care lucrează cu *grupuri* se va defini un grup, în listele de preferințe, prin intermediul tagului „<group>”.

Definim mai jos următoarea listă de preferințe $P(f_4)$: $\langle w_1, w_2 \rangle, w_1, w_2$:

```
<preferences name="f4">
  <element name="w1w2" level="1">
    <group> w1, w2 </group>
  </element>
  <element name="w1" level="2">
  <element name="w2" level="3">
</preferences>
```

Definirea unităților

Pentru *Stable-Allocation* este necesar să se specifice numărul de *unități* disponibile pentru fiecare element, deci acest lucru se va face în momentul definirii seturilor. Specificarea *max-unităților*, dintre două elemente, se va face în momentul specificării listelor de preferințe, în dreptul elementului în cauză.

În continuare vom defini două seturi și listele de preferințe ale elementelor $F = \{f\}$, $W = \{w\}$, $U(f) = 70$, $U(w) = 20$, $U(w, f) = 15$:

```
<set name="firms" size="1">
  <element name="f" units="70"/>
</set>
<set name="workers" size="1">
  <element name="w" units="20"/>
</set>

<preferences name="f">
  <element name="w" level="1" maxunits="15"/>
</set>

<preferences name="w">
  <element name="f" level="1" maxunits="15"/>
</set>
```

Restricții: max-unitățile trebuie specificate pentru fiecare „pereche” de elemente în parte, iar pentru aceeași pereche valoarea trebuie să fie identică, altfel instanța va fi considerate invalidă.

2.3 Specificații importante

În continuare se vor specifica câteva reguli de construire a unui XML valid:

- Tipul problemei trebuie să corespundă cu una din variantele prezentate în secțiunea 2.1
- Fiecare instanță trebuie să aibă definite minim 2 seturi de elemente
- Un set trebuie să conțină minim un element
- Numele elementelor trebuie să fie diferite
- Seturile se definesc primele
- Atributele care trebuie să primească ca și valoare un număr trebuie să fie obligatoriu definite pozitiv
- Toate elementele specificate la preferințe și în grupuri trebuie să fi fost definite în setul de elemente
- Un cuplu este definit sub tagul “<group>” și trebuie să conțină maxim două elemente
- Elementele dintr-un cuplu nu trebuie să aibe specificate decât o listă de preferințe pentru oricare dintre ele
- Pentru o instanță de tip **SA** este obligatoriu să fie definite unitățile pentru fiecare element în parte
- Pentru o instanță de tip **HR/MM** este necesar să se specific capacitățile elementelor

3 Arhitectură și implementare

În acest capitol se va discuta despre arhitectura proiectului cât și detaliile la nivel de implementare al cadrului de lucru.

Proiectul poate fi utilizat sub două forme. În primul rând, acesta poate fi utilizat ca o bibliotecă Java importată într-un proiect. De asemenea, având în vedere ușurința pe care o aduc în ultima perioadă microserviciile, proiectul prezintă și un serviciu REST prin care oferă soluții problemelor de *Stable-Matching*.

Cadrul de lucru prezintă patru niveluri importante:

- ❖ Interpretarea inputului, validarea și instanțierea
- ❖ Rezolvarea problemei
- ❖ Construirea rezultatului
- ❖ Serviciul REST

3.1 Schema generală a funcționării proiectului

Pentru oferirea unei imagini de ansamblu al serviciului REST pe care l-am creat, putem prezenta următoare schemă de interacționare cu API-ul:

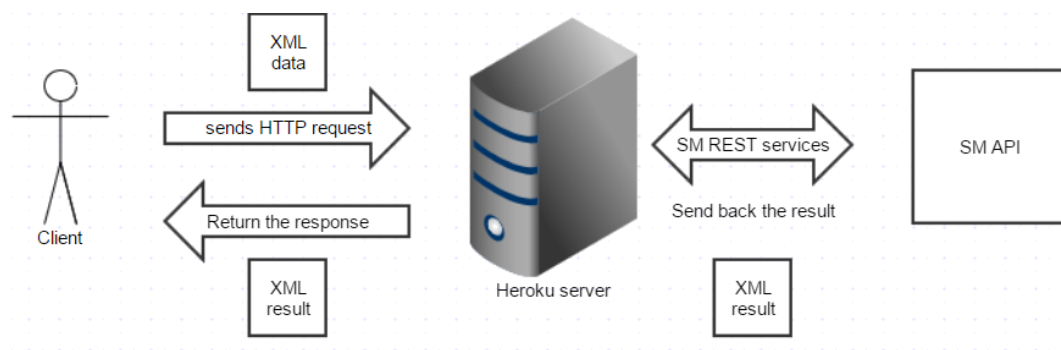


Fig. 4.1 Diagrama de interacțiune cu serviciul REST

- Serverul primește cererea pe care o trimite aplicației specificate (*smrestservices*)
- API-ul primește cererea, interpretează inputul, îl procesează și rezolvă problema, iar la final construiește rezultatul pe care urmează să îl trimită, prin intermediul protocolului HTTP, clientului, sub formă de XML

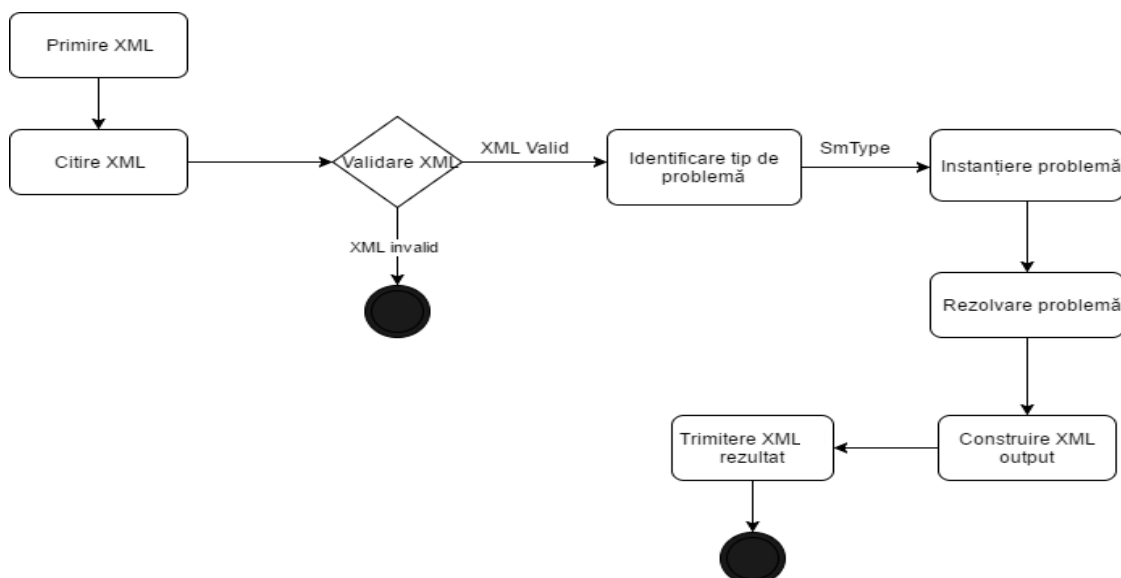


Fig 4.2 Fluxul activității în API după ce se primește XML-ul de input

3.2 Structura proiectului

În modulul de interpretare are loc parsarea XML-ului și verificarea tuturor condițiilor pentru a nu trimite date invalide modulului care se ocupă cu rezolvarea problemelor. În secțiunea 0

Modelarea problemelor de Stable-Matching prin reprezentări XML. Instanțierea problemelor au fost definite tiparele tuturor tipurilor de XML pentru problemele propuse. După parsare, are loc instanțierea. Diagrama de clase care modelează instanțele este reprezentată astfel:

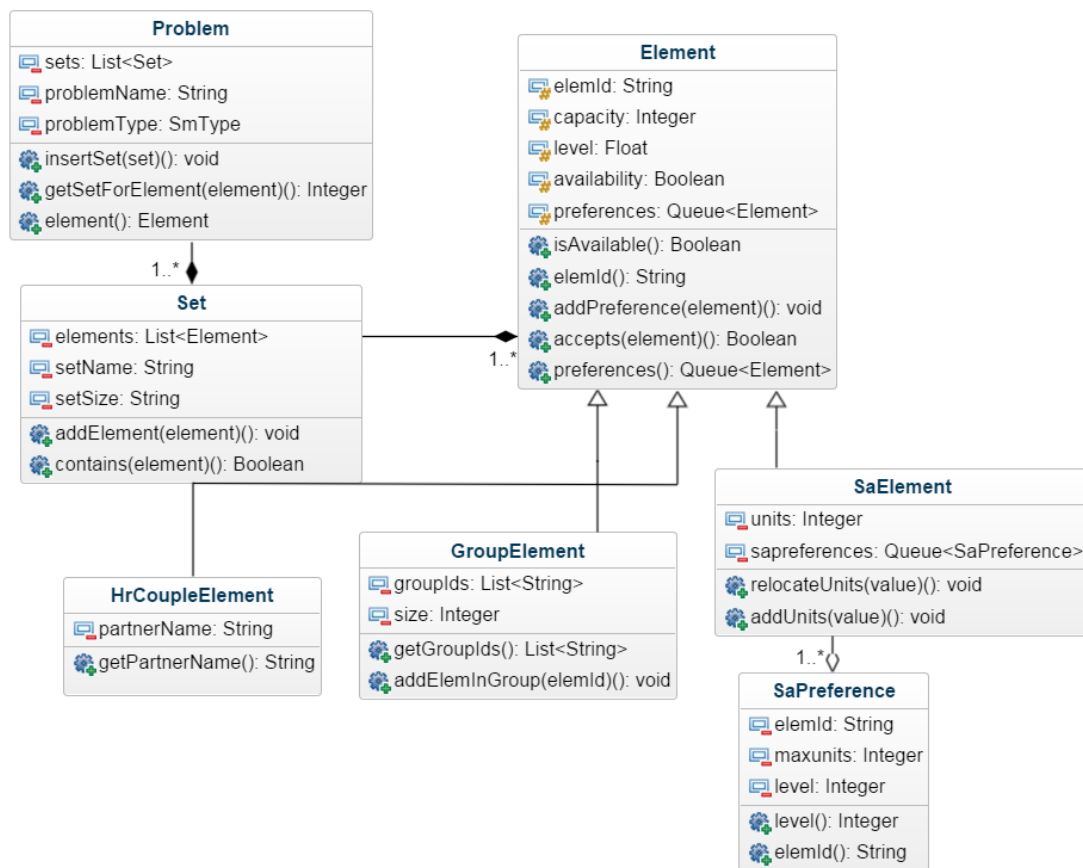


Fig 4.3 Clasele principale din pachetul “model”. Instanțierea problemei

Pentru rezolvarea problemei, modulul principal, anume nucleul a fost construit respectând ca și design pattern *Strategy*.

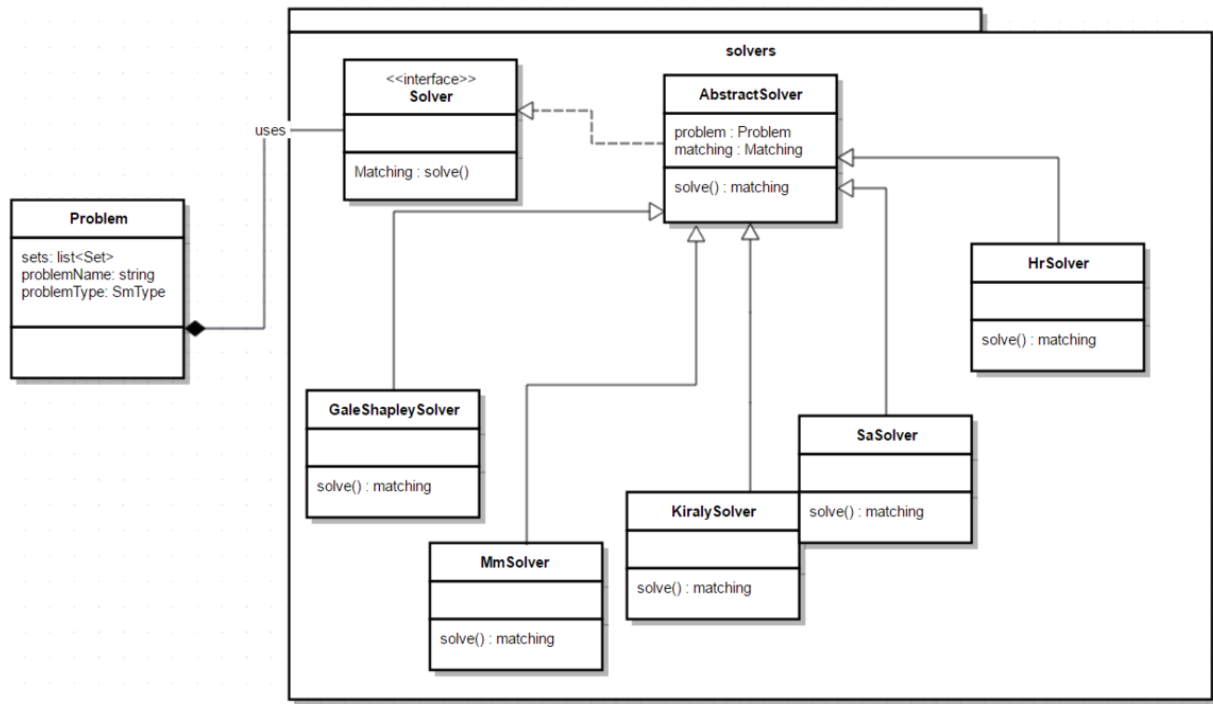


Fig 4.4 Clasele principale din pachetul “solvers”. Strategy Pattern

În cazul de față, *Problem* reprezintă contextul, ea conține datele pentru care se va ști ce tip de *Solver* este necesar. Bineînțeles, *SmType* este element esențial în determinarea tipului de problemă. *SmType* este definit ca fiind o enumerație astfel¹³:

```

package sm.utils.model;

public enum StableMatchingType {
    SM,
    SA,
    HR,
    MM
}

```

¹³ Tipurile sunt prezentate și în secțiunea 0

De asemenea, în pachetul “solvers” există și o “fabrică” de “solver”-e, anume SolverFactory, implementată astfel:

```
package sm.solvers;

import sm.utils.model.Problem;
import sm.utils.model.StableMatchingType;

public class SolverFactory {

    public static AbstractSolver getSolver(StableMatchingType problemType, Problem problem) {
        switch (problemType) {
            case SM:
                return new KiralySolver(problem);
            case HR:
                return new HrGaleShapleySolver(problem);
            case MM:
                return new MmSolver(problem);
            case SA:
                return new SaSolver(problem);
            default:
                return new GaleShapleySolver(problem);
        }
    }
}
```

Fig 4.5 clasa SolverFactory. Factory Method Pattern

Structura de date definită pentru reprezentarea rezultatului final poartă numele de *Matching* și prezentăm punctele principale din componența acesteia:

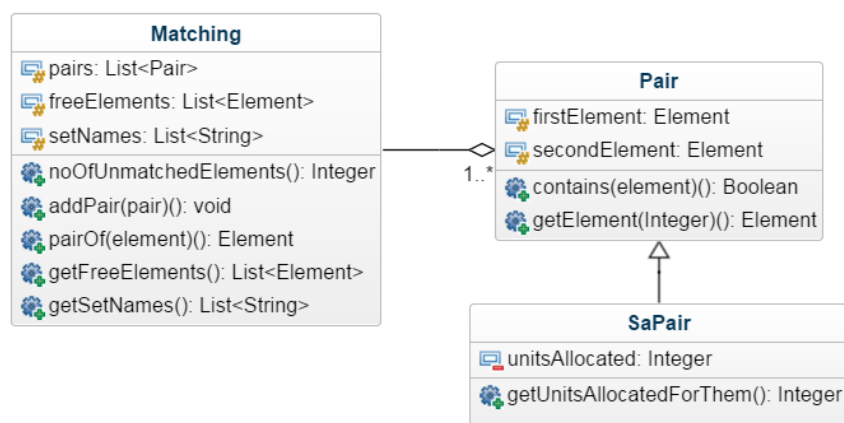


Fig 4.6 Clasele principale pentru reprezentarea rezultatului

3.3 Detalii de implementare al algoritmilor

```

@Override
public Matching solve() {
    SmLogger.start(true, algorithmName, solverName);

    matching = new Matching(problem.getSets().get(0).getElements());
    Set men = problem.getSets().get(0);

    /* array which holds a boolean that specifies if a man was already re-promoted */
    boolean visited[] = new boolean[problem.getSets().get(0).getSize()];
    Arrays.fill(visited, false);

    boolean finished = false;
    while(!finished){
        finished = true;
        matching = gsSolver.solve();
        int size = men.getSize();

        /* loop through all men */
        for (int i = 0; i < size; i++){
            Element man = men.getElements().poll();

            /* if the current man has already been promoted or it has a matching we pass over him */
            if (visited[i] || !matching.isFree(man))
                continue;

            /* we promote him */
            gsSolver.matching.addFreeElement(problem.element(man.elemId()));
            visited[i] = true;
            finished = false;
            Set women = problem.getSets().get(1);
            int womenSize = women.getSize();

            /* loop through women to raise the current man's priority with the value of EPSILON */
            for (int j = 0; j < womenSize; j++){
                Element woman = women.getElements().poll();

                /* we have to verify if the woman has the current man on her list, and if she does, his priority is raised with EPSILON */
                if (woman.accepts(man))
                    woman.prioritize(man, EPSILON);
            }
        }
        SmLogger.start(false, algorithmName, solverName);
        return matching;
    }
}

```

Fig 4.7 Implementarea algoritmului propus de Kiraly

Am preferat să prezint algoritmul Király deoarece este unul din cei mai potriviți pentru a înfățișa o imagine de bază a structurii unui algoritm de *Stable-Matching*. După cum a fost prezentat deja în capitolul 1, el pornește de la un matching construit, deci, folosește algoritmul GS pentru a construi distribuția.

3.4 Serviciul REST

Unele servicii sunt mult mai ușor de folosit și de menținut la zi cu toate modificările dacă prezintă și o interfață web, de aceea am optat și pentru oferirea unui serviciu REST.

Pentru implementarea serviciului REST am folosit SpringBoot care oferă o modalitate practică și ușoară. Cea mai importantă rută este cea care rezolvă instanța primită, anume ruta de POST.

```
@RequestMapping(value = "/algorithm/{param}", method = RequestMethod.POST, consumes = MediaType.APPLICATION_XML_VALUE,
    headers = "Accept=application/xml", produces = MediaType.APPLICATION_XML_VALUE)
public ResponseEntity<?> getMatching(@PathVariable("param") String param, @RequestBody String document){
    log.info("New request on /algorithm/" + param);

    SMSService service = new SMSService();
    ResponseXml xml = service.manage(param, document);

    if (xml instanceof SolvedXml)
        return new ResponseEntity<>(xml, HttpStatus.OK);
    else return new ResponseEntity<>(xml, HttpStatus.BAD_REQUEST);
}
```

Fig 4.8 Ruta POST pentru apelarea serviciului de *Stable-Matching*

În caz de XML-ul primit nu trece de procesul de validare, nu sunt respectate cerințele specificate în 2.3, se va întoarce un XML de eroare. La generarea lui s-a folosit JAXB. Spre exemplu, clasa de bază, adică *ResponseXml* prezintă următoarea implementare:

```
package rest.model;

import ...

@XmlRootElement(name = "result")
public class ResponseXml {
    protected String algorithm;

    @XmlAttribute(name = "algorithm")
    public String getAlgorithm() { return algorithm; }

    public void setAlgorithm(String algorithm) { this.algorithm = algorithm; }
}
```

Fig 4.9 ResponseXml, implementare cu JAXB

Modele de output sunt prezentate în secțiunea 5. Structura claselor implicate în construirea XML-ului ce urmează a fi returnat este următoarea:

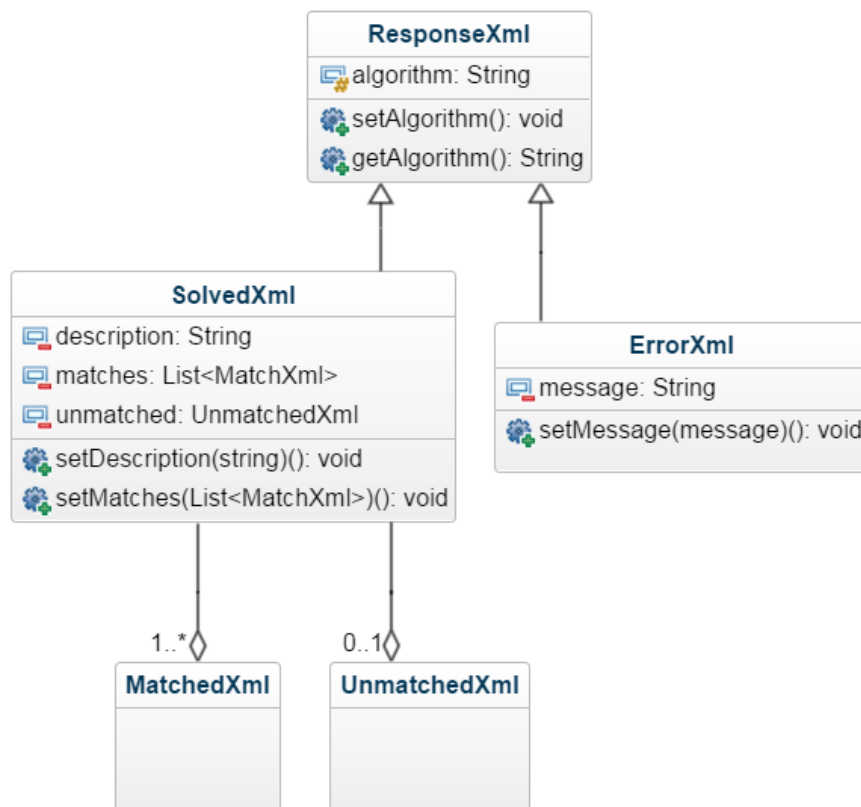


Fig 4.10 Structura generală a claselor care modelează XML-ul ce va fi returnat

3.5 Tehnologii și platforme utilizate

Ca și tehnologie principală am optat pentru Java, aceasta oferind un mod ușor de implementare al structurii proiectului și prezentând concept care permit gestionarea într-un mod eficient al acestuia. De asemenea, Java are și o varietate de structuri de date ușor de utilizat în cadrul unor algoritmi.

Pentru crearea serviciului REST, am folosit framework-ul **SpringBoot**. REST¹⁴ este o arhitectură înclinată spre expunerea unui mod uniform și ușor de transfer de date [11]. Acest lucru se realizează prin cereri și răspunsuri, metode HTTP¹⁵ ¹⁶.

Cheia care stă la baza acestei arhitecturi o reprezintă modul de interpretare și reprezentare al resurselor. REST oferă o idee nouă prin care separă complet interfața unei aplicații de partea de server, astfel reușind să fie extrem de ușor de creat mai multe aplicații care consumă același serviciu. Ca și imagine de ansamblu, prin servicii REST se înțelege o arhitectura care, prin intermediul URI¹⁷-urilor, expune o multitudine de resurse și servicii.

SpringBoot oferă un mod ușor de lucru cu serviciile REST, acesta realizându-se prin adnotări, cum ar fi:

- **@Controller** => adnotare care desemnează o clasă ca fiind un controller ce va intercepta acțiunile ce vor fi cerute pentru o anumită resursă
- **@RequestMapping("/sm")** => adnotare prin care se asigură că toate cererile către "/sm" vor fi mapate către o metodă/clasă; în cazul nostrum este vorba de o clasă, anume controller-ul `SmServiceController`

Cea de-a doua adnotare prezintă și o multitudine de attribute care ajută la specificarea tuturor informațiilor necesare pentru o anumită ruta, spre exemplu: *value*, *method*, *consumes* (opțional), *produces* (opțional), *headers* etc. Utilizarea acestora poate fi observată în figura 4.8.

La realizarea XML-urilor, pentru a putea transpune datele din obiecte în noduri XML am utilizat JAXB. Ca și SpringBoot, JAXB lucrează tot cu ajutorul adnotărilor prin care oferă o modalitate de mapare a obiectelor. Câteva adnotări importante ar fi: `@RootElement`, `@XmlAttribute`, `@XmlElement`. Un exemplu în care au fost utilizate se poate constata în figura 4.9.

¹⁴ Representational State Transfer în eng.

¹⁵ Hypertext Transfer Protocol

¹⁶ HTTP verbs în eng.

¹⁷ Uniform Resource Identifiers în eng

Pentru “găzduirea”¹⁸ serviciului construit am folosit un server gratuit, Heroku, acesta punând la dispoziție fiecărui utilizator această facilități într-o anumită măsură.

Heroku este o platformă în Cloud care are la bază un container de sistem administrat cu servicii de date integrate și cu un ecosistem puternic, acesta ajutând la facilitarea lansării¹⁹ și a rulării aplicațiilor pe web [12].

4 Lucrul cu serviciul de *Stable-Matching*

Cum am menționat și anterior, serviciul poate fi folosit în două moduri, anume:

- ❖ Ca și cadru de lucru, importând biblioteca în propriul proiect
- ❖ Folosind serviciul REST

Utilizarea serviciului REST

În continuare se va prezenta modul în care se vor face cereri către serviciul rest. Ruta destinată rezolvării problemelor se apelează printr-o cerere²⁰ de tip POST la URI-ul:

<http://smrestservices.herokuapp.com/sm/algorithm/{param}>.

{param} reprezintă tipul problemei, el trebuie să fie identic cu numele problemei specificat în XML-ul trimis prin POST. Rezultatul pe care clientul îl va primi, ca urmare al cererii, va fi tot un XML. Un exemplu ar putea fi:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<result description="Matching result for the instance proposed" algorithm="Gale
Shapley for one-to-many instance">
  <match>
    <element name="r1" set="residents"/>
    <element name="h1" set="hospitals"/>
  </match>
  <match>
    <element name="r2" set="residents"/>
    <element name="h1" set="hospitals"/>
  </match>
  <unmatched>
    <element name="r3" set="residents"/>
  </unmatched>
</result>
```

¹⁸ host în eng.

¹⁹ deploy în eng.

²⁰ request HTTP în eng.

```

<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Studio 2017 - Developer Bundle Edition (Trial) 15.1.4.7515
(https://www.liquid-technologies.com) -->
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="unbounded" name="match">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="element"
type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element minOccurs="1" name="unmatched">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="element"
type="xs:string" />
            </xs:sequence>
            <xs:attribute name="number" type="xs:unsignedByte" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="algorithm" type="xs:string" use="required" />
      <xs:attribute name="description" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

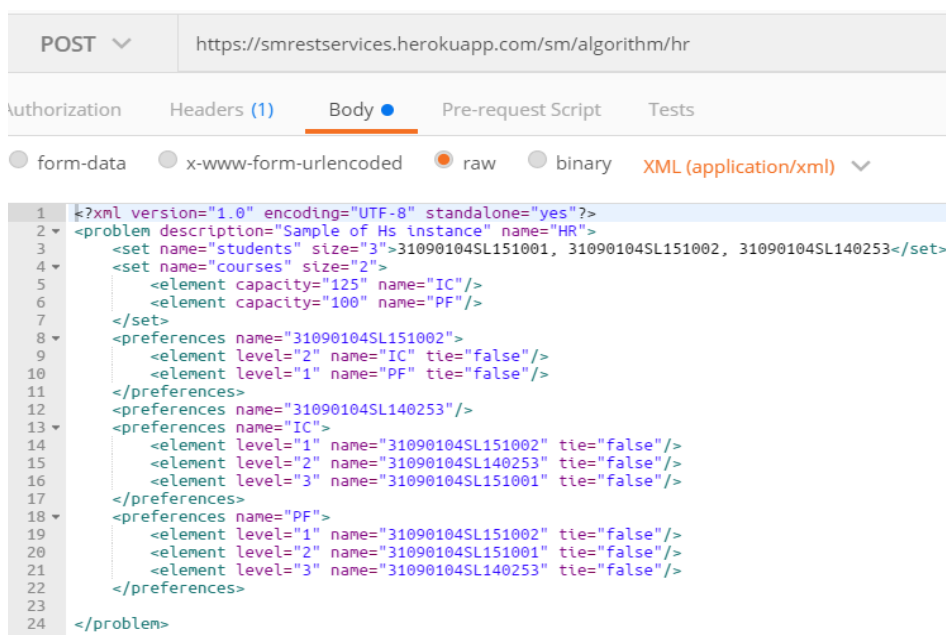
Schema XSD a modelului de output

După cum este menționat în această schemă și după cum am observat și în exemplu, output-ul este alcătuit din:

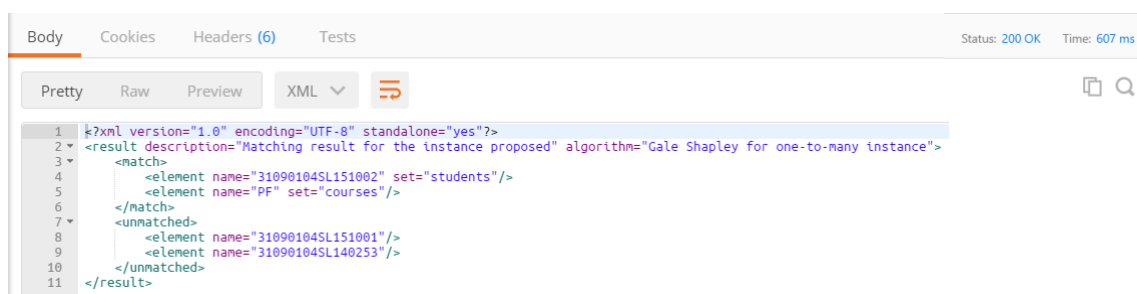
- Nodul rădăcină care poartă numele **<result>** și prezintă două atribute care oferă detalii legate de modul în care a fost generat acest XML rezultat, prin **"algorithm"** se specifică numele algoritmului folosit, iar prin **"description"** se descrie în câteva cuvinte tipul acestuia
- Între 0 și n elemente **<match>** care prezintă câte o pereche formată din cel puțin două noduri **<element>**
- Fiecare **<element>** are precizat numele și setul din care face parte

- Prin tagul **<unmatched>** sunt prezentate acele elemente (din primul set, adică setul care face “propunerii”) care nu au reușit să fie distribuite cu nicio altă entitate din setul opus

În imaginea de mai jos este afișat un exemplu de cerere POST către serviciu:



Iar răspunsul primit în urma cererii va fi de tipul²¹:



²¹ Interfața folosită pentru clientul care apelează serviciul este POSTMAN (<https://www.getpostman.com/apps>)

Acum se va oferi un exemplu de rezultat în cazul în care se trimite un XML invalid. Spre exemplu vom specifica o dimensiune invalid pentru unul dintre cele două seturi.

Vom trimite XML-ul:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<problem description="Sample of HR instance" name="HR">
  <set name="students" size="3">31090104SL151001, 31090104SL151002,
31090104SL140253</set>
  <set name="courses" size="-1">
    <element capacity="125" name="IC"/>
    <element capacity="100" name="PF"/>
  </set>
  <preferences name="31090104SL151002">
    <element level="2" name="IC"/>
    <element level="1" name="PF"/>
  </preferences>
  <preferences name="31090104SL140253"/>
  <preferences name="IC">
    <element level="1" name="31090104SL151002"/>
    <element level="2" name="31090104SL140253"/>
    <element level="3" name="31090104SL151001"/>
  </preferences>
  <preferences name="PF">
    <element level="1" name="31090104SL151002"/>
    <element level="2" name="31090104SL151001"/>
    <element level="3" name="31090104SL140253"/>
  </preferences>
</problem>
```

Răspunsul care se primește în urma creării unei cereri cu acest XML este:



5 Concluzii și perspective de viitor

Cadrul de lucru prezentat în lucrarea de față pune la dispoziție un serviciu prin care se rezolvă diferite tipuri de probleme de *Stable-Matching*. Aceste probleme se întâlnesc în situații din viața reală oricând se vorbește despre distribuirea unui eșantion de entități în diferite circumstanțe, spre exemplu distribuția elevilor la un liceu, repartizarea unor profesori la mai multe licee/școli etc.

Există numeroase variante ale acestor probleme, în lucrarea de față prezentându-se cele mai generale tipuri care s-ar găsi în viața reală și ar fi utile pentru utilizarea în situații precum cele specificate anterior. Unele probleme destul de particulare care s-ar putea adăuga ar fi:

- ***Students/Projects Allocation*** – un anumit număr de studenți doresc să se înscrie la examenul de licență și ar trebui să își aleagă un proiect; un profesor poate avea unul sau mai multe proiecte și prezintă un număr limitat de locuri, de asemenea și proiectele prezintă un anumit număr de locuri disponibile
- ***Stable Roommates*** – este o problemă de *Stable-Matching* aparte, distribuția se face între elemente din același set, aici fiind prezent un singur set
- ***Stable Matchings 3D*** – o serie de probleme care încearcă să facă distribuția între trei seturi de entități (ex: femei, bărbați și câini)

De asemenea, în diferite lucrări specifice acestui subiect, s-a demonstrat faptul că nu există doar un singur *Stable-Matching* pentru o instanță, există mai multe, unele chiar neavând caracteristica de optimalitate într-un singur sens (*woman-optimal/men-optimal* etc). Deci s-ar putea construi algoritmi care să găsească întreg setul de matching-uri și să ofere anumite scoruri acestora.

6 Bibliografie

- [1] D. Gale and L. S. Shapley, “College admissions and the stability of marriage”, *Amer. Math. Monthly*, Vol.69, pp. 9–15, 1962.
- [2] M. Baïou and M. Balinski, “Many-to-many matching: stable polyandrous polygamy (or polygamous polyandry)”, *Discrete Applied Mathematics*, Vol. 101, pp. 1–12, 2000.
- [3] Kazuo Iwama, “A Survey of the Stable Marriage Problem and Its Variants”, *International Conference on Informatics Education and Research for Knowledge-Circulating Society*, 2008.
- [4] R. W. Irving, D. F. Manlove, “Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems”, *Journal of Combinatorial Optimization* (2007)
- [5] Zoltán Király, “Better and simpler approximation algorithms for the stable marriage problem”, *Egerváry Research Group on Combinatorial Optimization*, April 4, 2008.
- [6] Robert W. Irving, David F. Manlove and Sandy Scott, “The Hospitals/Residents Problem with Ties”.
- [7] Ruth Martínez, Jordi Massó, Alejandro Neme and Jorge Oviedo, “An algorithm to compute the full set of many-to-many stable matchings”, *Journal of Economic Literature Classification Number: C78*, July, 2003.
- [8] Robert W. Irving, David F. Manlove and Sandy Scott, “Strong Stability in the Hospitals/Residents Problem”.
- [9] David F. Manlove, Robert W. Irving, Kazuo Iwamay, Shuichi Miyazaki, and Yasufumi Morita, “Hard Variants of Stable Marriage”.
- [10] Dan Gusfiel, “Three fast algorithms for four problems in Stable Marriage”, *Society for Industrial and Applied Mathematics, Siam J. Comput.*, Vol. 16, No.1, February, 1987.

[11] Robert W. Irving, “Stable marriage and indifference”, *Discrete Applied Mathematics* 48 (1994) 261–272, 5 February, 1990.

[12] Organising Committee of the Third International Competition of CSP Solvers, “XML Representation of Constraint Networks Format XCSP 2.1”, 15 January, 2008.

[13] Liquid Studio 2017

[14] <https://spring.io>

[15] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[16] <https://www.heroku.com/platform>

[17] <https://www.ams.org/samplings/feature-column/fc-2015-03>

[18] Pentru diagrame:

<https://www.draw.io>

<https://www.genmymodel.com>

<https://www.gliffy.com>

Anexa A

Model de XML pentru o instanță de Stable-Marriage

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name = "SM" dimension = "2D" description = "Sample of a SM
instance">
  <set name = "women" size = "3">
    w1, w2, w3
  </set>
  <set name = "men" size = "3">
    m1, m2, m3
  </set>
  <preferences name = "w1">
    <element name = "m1" level = "1" />
    <element name = "m2" level = "2" />
    <element name = "m3" level = "3" />
  </preferences>

  <preferences name = "w2">
    <element name = "m2" level = "1" />
    <element name = "m1" level = "2" />
    <element name = "m3" level = "3" />
  </preferences>

  <preferences name = "w3">
    <element name = "m2" level = "1" />
    <element name = "m1" level = "2" />
    <element name = "m3" level = "3" />
  </preferences>

  <preferences name = "m1">
    <element name = "w1" level = "1" />
    <element name = "w2" level = "2" />
  </preferences>

  <preferences name = "m2">
    <element name = "w1" level = "1" />
    <element level = "2" tie = "true"> w3, w2 </element>
  </preferences>

  <preferences name = "m3">
    <element name = "w1" level = "1" />
    <element name = "w2" level = "2" />
  </preferences>
</problem>
```

Anexa B

Model de XML pentru o instanță de Hostpitals/Residents

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name = "HR" category = "" dimension = "2D" description =
"Sample of a HR instance">

  <set name = "residents" size = "3">
    r1, r2, r3
  </set>
  <set name = "hospitals" size = "2">
    <element name = "h1" capacity = "4" />
    <element name = "h2" capacity = "2" />
  </set>

  <group>
    <couple> r1, r3 </couple>
  </group>

  <preferences name = "r2">
    <element name = "h2" level = "1" />
    <element name = "h1" level = "2" />
  </preferences>

  <preferences name = "r3">
    <element level = "1" tie = "true"> h1, h2 </element>
  </preferences>

  <preferences name = "h1">
    <element name = "r1" level = "1" />
    <element name = "r2" level = "2" />
    <element name = "r3" level = "3" />
  </preferences>

  <preferences name = "h2">
    <element level = "1" tie = "true"> r3, r2 </element>
    <element name = "r1" level = "4" />
  </preferences>

</problem>
```

Anexa C

Model de XML pentru o instanță many-to-many (Teachers Repartition)

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name = "MM" dimension = "2D" description = "Sample of a MM instance">
  <set name = "firms" size = "4">
    <element name = "f1" capacity = "2" />
    <element name = "f2" capacity = "2" />
    <element name = "f3" capacity = "2" />
    <element name = "f4" capacity = "2" />
  </set>
  <set name = "workers" size = "4">
    <element name = "w1" capacity = "2" />
    <element name = "w2" capacity = "2" />
    <element name = "w3" capacity = "2" />
    <element name = "w4" capacity = "2" />
  </set>
  <preferences name = "f1">
    <element name = "w1w2" level = "1">
      <group> w1, w2</group>
    </element>

    <element level = "2" tie = "true">
      <group name = "w1w3"> w1, w3 </group>
      <group name = "w3w4"> w3, w4 </group>
    </element>

    <element name = "w2w4" level = "3">
      <group> w2, w4 </group>
    </element>

    <element name = "w1w4" level = "4">
      <group> w1, w4 </group>
    </element>

    <element name = "w2w3" level = "5">
      <group> w2, w3 </group>
    </element>
  </preferences>
  <preferences name = "f2">
    <element level = "1" tie = "true">
      <group name = "w1w2"> w1, w2 </group>
      <group name = "w1w4"> w1, w4 </group>
    </element>

    <element level = "2" tie = "true">
      <group name = "w2w3"> w2, w3 </group>
      <group name = "w2w4"> w2, w4 </group>
    </element>
  </preferences>
</problem>
```

```

    <element name = "w3w4" level = "3">
      <group> w3, w4 </group>
    </element>

    <element name = "w1w3" level = "4">
      <group> w1, w3 </group>
    </element>
  </preferences>
  <preferences name = "f3">
    <element name = "w1w4" level = "1">
      <group> w1, w4 </group>
    </element>

    <element name = "w1w2" level = "2">
      <group> w1, w2 </group>
    </element>

    <element name = "w2w4" level = "3">
      <group> w2, w4 </group>
    </element>

    <element name = "w1" level = "4" />
    <element name = "w2" level = "5" />
    <element name = "w4" level = "6" />
  </preferences>
  <preferences name = "f4">
    <element name = "w1w2" level = "1">
      <group> w1, w2 </group>
    </element>

    <element name = "w1" level = "2" />
    <element name = "w2" level = "3" />
  </preferences>
  <preferences name = "w1">
    <element name = "f3f4" level = "1">
      <group> f3, f4 </group>
    </element>

    <element level = "2" tie = "true">
      <group name = "f2f3"> f2, f3 </group>
      <group name = "f2f4"> f2, f4 </group>
    </element>

    <element name = "f1f3" level = "3">
      <group> f1, f3 </group>
    </element>

    <element name = "f1f2" level = "4">
      <group> f1, f2 </group>
    </element>

    <element name = "f1" level = "5" />
    <element name = "f2" level = "6" />

```

```

    <element name = "f3" level = "7" />
    <element name = "f4" level = "8" />
</preferences>
<preferences name = "w2">
  <element level = "1" tie = "true">
    <group name = "f3f4"> f3, f4 </group>
    <group name = "f2f4"> f2, f4 </group>
  </element>

  <element name = "f2f3" level = "2">
    <group name = "f2f3"> f2, f3 </group>
  </element>

  <element name = "f1f4" level = "3">
    <group> f1, f4 </group>
  </element>

  <element level = "4" tie = "true">
    <group name = "f1f3"> f1, f3 </group>
    <group name = "f1f2"> f1, f2 </group>
  </element>

  <element name = "f1" level = "5" />
  <element name = "f2" level = "6" />
  <element name = "f3" level = "7" />
  <element name = "f4" level = "8" />
</preferences>
<preferences name = "w3">
  <element level = "1" tie = "false">
    <group name = "f1f2"> f1, f2 </group>
    <group name = "f2f4"> f2, f4 </group>
    <group name = "f1f3"> f1, f3 </group>
  </element>

  <element name = "f2f3" level = "2">
    <group> f2, f3 </group>
  </element>

  <element name = "f1f4" level = "3">
    <group> f1, f4 </group>
  </element>

  <element name = "f3f4" level = "4">
    <group> f3, f4 </group>
  </element>

  <element name = "f1" level = "5" />
  <element name = "f2" level = "6" />
  <element name = "f3" level = "7" />
  <element name = "f4" level = "8" />
</preferences>

```

```

<preferences name = "w4">
  <element level = "1" tie = "true">
    <group name = "f1f2"> f1, f2 </group>
    <group name = "f2f4"> f2, f4 </group>
    <group name = "f3f4"> f3, f4 </group>
  </element>

  <element name = "f1f3" level = "2">
    <group> f1, f3 </group>
  </element>

  <element name = "f1f4" level = "3">
    <group> f1, f4 </group>
  </element>

  <element name = "f2f3" level = "4">
    <group> f2, f3 </group>
  </element>

  <element name = "f1" level = "5" />
  <element name = "f2" level = "6" />
  <element name = "f3" level = "7" />
  <element name = "f4" level = "8" />
</preferences>
</problem>

```

Anexa D

Model de XML pentru o instanță de tip many-to-many (Stable-Allocation)

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name = "SA" dimension = "2D" description = "Sample of a SA instance">
  <set name = "workers" size = "7">
    <element name = "w1" units = "20" />
    <element name = "w2" units = "16" />
    <element name = "w3" units = "15" />
    <element name = "w4" units = "14" />
    <element name = "w5" units = "19" />
    <element name = "w6" units = "21" />
    <element name = "w7" units = "23" />
  </set>
  <set name = "firms" size = "3">
    <element name = "f1" units = "70" />
    <element name = "f2" units = "55" />
    <element name = "f3" units = "31" />
  </set>

  <preferences name = "f1">
    <element name = "w3" level = "1" maxunits = "15" />
    <element name = "w7" level = "2" maxunits = "15" />
    <element name = "w2" level = "3" maxunits = "15" />
    <element name = "w1" level = "4" maxunits = "15" />
    <element name = "w5" level = "5" maxunits = "15" />
    <element level = "6" maxunits = "15" tie = "true">
      w4, w6
    </element>
  </preferences>
  <preferences name = "f2">
    <element name = "w2" level = "1" maxunits = "12" />
    <element name = "w1" level = "2" maxunits = "12" />
    <element name = "w5" level = "3" maxunits = "12" />
    <element name = "w4" level = "4" maxunits = "12" />
    <element name = "w3" level = "5" maxunits = "12" />
    <element name = "w6" level = "6" maxunits = "12" />
  </preferences>
  <preferences name = "f3">
    <element name = "w2" level = "1" maxunits = "10" />
    <element name = "w7" level = "2" maxunits = "10" />
    <element name = "w4" level = "3" maxunits = "10" />
    <element name = "w5" level = "4" maxunits = "10" />
    <element name = "w3" level = "5" maxunits = "10" />
    <element name = "w6" level = "6" maxunits = "10" />
    <element name = "w1" level = "6" maxunits = "10" />
  </preferences>
  <preferences name = "w1">
    <element name = "f2" level = "1" maxunits = "12" />
  </preferences>
</problem>
```

```

        <element name = "f3" level = "2" maxunits = "10" />
        <element name = "f1" level = "3" maxunits = "15" />
    </preferences>

    <preferences name = "w2">
        <element name = "f2" level = "1" maxunits = "12" />
        <element name = "f1" level = "2" maxunits = "15" />
        <element name = "f3" level = "3" maxunits = "10" />
    </preferences>

    <preferences name = "w3">
        <element name = "f1" level = "1" maxunits = "15" />
        <element name = "f2" level = "2" maxunits = "12" />
    </preferences>

    <preferences name = "w4">
        <element name = "f2" level = "1" maxunits = "12" />
        <element name = "f3" level = "2" maxunits = "10" />
        <element name = "f1" level = "3" maxunits = "15" />
    </preferences>
    <preferences name = "w5">
        <element name = "f1" level = "1" maxunits = "15" />
        <element name = "f3" level = "2" maxunits = "10" />
        <element name = "f2" level = "3" maxunits = "12" />
    </preferences>

    <preferences name = "w6">
        <element name = "f3" level = "1" maxunits = "10" />
        <element name = "f2" level = "2" maxunits = "12" />
        <element name = "f1" level = "3" maxunits = "15" />
    </preferences>

    <preferences name = "w7">
        <element name = "f1" level = "1" maxunits = "15" />
        <element name = "f2" level = "2" maxunits = "12" />
        <element name = "f3" level = "3" maxunits = "10" />
    </preferences>
</problem>

```