A project report on

PROFESSOR-COURSE ALLOTMENT RECOMMENDER SYSTEM

By

Dua, Manpriya (G00950511)
Jammula, Rakesh (G00913614)
Monga, Bhawna (G00921287)

Decision Guidance Systems (CS787)

Department of Computer Science

Volgenau School of Engineering

PROBLEM STATEMENT

Currently, the allotment of courses to professors (who would teach them in the upcoming semesters) is done manually in many schools and Universities. This manual allocation is done based on two main conditions:

- The availability of the instructor
- Instructor's expertise to take that particular class.

While doing so, it is difficult to satisfy the preference of various professors and when multiple professors show interesting in teaching a particular class, the allotment becomes all the more complex and difficult.

WHAT CAN BE DONE

Instead of the manual allotment, we could automate the process and make a system that allots courses by taking into consideration the Professor satisfaction.

We achieve this, by taking a list of preferences for each professor as an input JSON structure along with the available professors and courses.

The details about the complete input list and flow of the process is mentioned in the Project Implementation section. At the end we, our goal is to try and maximize the overall Professor Satisfaction and its Overall Fairness.

PROJECT IMPLEMENTATION

PROJECT REQUIREMENTS

We used the following software/languages to achieve our goal:

Languages:

- JSONiq: It is a query and processing language specifically designed for the popular JSON data model. It is an expressive and highly optimizable language to query and update any kind of JSONiq store or resource.
- Decision Guidance Analytical Language (DGAL): It allows the creation of modular, reusable and composable models which are stored in the analytical knowledge base independently of the tasks and tools that use them. DGAL helps in optimizing the decision variable based on the constraints put in place.

Software: Atom and GMU Unity Server (DGAL)

Hardware: Laptop

INPUT

Below is the list of JSON structures that we have taken in as the input:

- Professors and Classes they can teach
- Courses to be offered per semester for the next 4 semesters.
- List of all core courses that can be offered
- Professor and their preferences per semester, for next 4 semesters

By using the input structures Professors and Classes they can teach, and Courses to be offered per semester, a JSONIQ query is used to get one more input structure. The new structure consists of all the Course: Prof, pairs who can teach that course, in that semester. A decision variable flag will be appended to each such structure. The decision variable flag will be set to 0 or 1 depending on the constraints and the optimization of the allocation.

1. professors.jq: This JSON structure contains the id of the Professor (pid), the name of the professor (pname) and an array of the courses he/she can teach (can_teach)

professors.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/professors.jq";

declare variable $ns:professors := (
    { "pid": 14,
      "pname": "Brodsky",
      "can_teach": ["CS550", "CS787", "INFS740", "CS650" ]
    },
    { "pid": 15,
      "pname": "Tecuci",
      "can_teach": ["CS681","CS583","CS580", "CS685"]
    },
    { "pid": 16,
      "pname": "Wechsler",
      "can_teach": ["CS580", "CS682", "CS773", "SWE645"]
    },
    { "pid": 17,
      "pname": "Dana",
      "can_teach": ["CS583", "CS685", "CS773","CS795"]
    });

```

2. courses.jq: This JSON structure contains the semester number (sem) and an array of the courses that are planned to be taught in that semester (courses).

courses.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/courses.jq";

declare variable $ns:courses := (
    { sem: 1, courses: ["CS550", "INFS740" ,"CS681", "CS685", "CS773", "CS583","SWE645","CS795"]},
    { sem: 2, courses: ["CS583", "SWE645", "CS550", "CS650", "CS685", "CS580","CS681","CS773"]},
    { sem: 3, courses: ["CS650", "CS580", "CS787", "CS681", "CS682", "SWE645","CS583","CS685"]},
    { sem: 4, courses: ["INFS740", "CS550", "CS685", "CS580","CS773", "CS682","CS583","CS795"]});

```

3.professors_prefernece.jq: This JSON structure contains the semester number (sem) and a Preference list of courses for each Professor(pref1 and pref2) and the number of minimum (min) and maximum (max) courses the professor can teach in that particular semester

professors_preference.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/professors_preference.jq";

declare variable $ns:professors_preference := (
{
  "sem":1,
  "Brodsky"   : {"pref1":"CS550" , "pref2":"INFS740", "min":1, "max":2},
  "Tecuci"    : {"pref1":"CS685" , "pref2":"CS681"  , "min":1, "max":3},
  "Wechsler" : {"pref1":"SWE645", "pref2":"CS773"  , "min":1, "max":2},
  "Dana"      : {"pref1":"CS773" , "pref2":"CS583"  , "min":1, "max":3}
},
{
  "sem":2,
  "Brodsky"   : {"pref1":"CS550" , "pref2":"CS650"  , "min":1, "max":2},
  "Tecuci"    : {"pref1":"CS685" , "pref2":"CS580"  , "min":1, "max":3},
  "Wechsler" : {"pref1":"CS773" , "pref2":"SWE645" , "min":1, "max":3},
  "Dana"      : {"pref1":"CS685" , "pref2":"CS583"  , "min":1, "max":2}
},
{
  "sem":3,
  "Brodsky"   : {"pref1":"CS787" , "pref2":"CS650"  , "min":1, "max":2},
  "Tecuci"    : {"pref1":"CS583" , "pref2":"CS580"  , "min":1, "max":3},
  "Wechsler" : {"pref1":"CS682" , "pref2":"SWE645" , "min":1, "max":2},
  "Dana"      : {"pref1":"CS583" , "pref2":"CS685"  , "min":1, "max":3}
},
{
  "sem":4,
  "Brodsky"   : {"pref1":"CS550" , "pref2":"INFS740", "min":1, "max":3},
  "Tecuci"    : {"pref1":"CS580" , "pref2":"CS685"  , "min":1, "max":2},
  "Wechsler" : {"pref1":"CS682" , "pref2":"CS580"  , "min":1, "max":2},
  "Dana"      : {"pref1":"CS583" , "pref2":"CS773"  , "min":1, "max":2}
});

```

4. ProfCoursePairs.jq: This JSON structure contains the decision variable, flag, on which we try to allocate courses to professors and in turn maximize professor satisfaction.

This structure consists of a list of all professor course pairs for a semester, such that the course can be taught by the Professor.

We have a flag variable set for each such professor and course pair, and it is set to 1 if the course has been allotted and 0 otherwise.

ProfCoursePairs.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/ProfCoursePairs.jq";

declare variable $ns:ProfCoursePairs:= (
    { "sem" : 1,
     "List" : [ { "course" : "CS550"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "INFS740", "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS681"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS773"   , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "SWE645" , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS773"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS795"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  } ] },

    { "sem" : 2,
     "List" : [ { "course" : "CS550"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS650"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS580"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS681"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "SWE645" , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS580"   , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS773"   , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS773"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  } ] },

    { "sem" : 3,
     "List" : [ { "course" : "CS650"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS787"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS580"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS681"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Tecuci"    , "flag" : {"integer?": 0}  },
                { "course" : "CS580"   , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS682"   , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "SWE645" , "Prof" : "Wechsler" , "flag" : {"integer?": 0}  },
                { "course" : "CS583"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  },
                { "course" : "CS685"   , "Prof" : "Dana"      , "flag" : {"integer?": 0}  } ] },

    { "sem" : 4,
     "List" : [ { "course" : "INFS740", "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
                { "course" : "CS550"   , "Prof" : "Brodsky"   , "flag" : {"integer?": 0}  },
```

5. core_courses.jq: This JSON structure contains a list of all the courses that are designated as core courses.

core_courses.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/core_courses.jq";

declare variable $ns:core_courses := (["CS583","CS550","CS580","ISA562","SWE619","CS581","CS584",
                                       "CS555","CS540","SWE621","CS551"]);

```

6. core_demand.jq: This JSON structure contains a list of the courses , the number of credit hours of the class, and the demand of class, i.e. the number of students that would register depending on the Professor teaching the class.

This structure is mainly used in calculating the metric Full Time Equivalent (FTE)

course_demand.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/course_demand.jq";

declare variable $ns:course_demand := (
{"courses":"CS550"  , "credit_hours":3, "demand":[ {"Prof":"Brodsky" , "No_of_Students":34} ] },
{"courses":"CS580"  , "credit_hours":3, "demand":[ {"Prof":"Tecuci"  , "No_of_Students":44},
                                                   {"Prof":"Wechsler", "No_of_Students":24} ] },
{"courses":"CS583"  , "credit_hours":3, "demand":[ {"Prof":"Dana"    , "No_of_Students":31},
                                                   {"Prof":"Tecuci"  , "No_of_Students":14} ] },
{"courses":"CS650"  , "credit_hours":3, "demand":[ {"Prof":"Brodsky" , "No_of_Students":44} ] },
{"courses":"CS681"  , "credit_hours":3, "demand":[ {"Prof":"Tecuci"  , "No_of_Students":39} ] },
{"courses":"CS682"  , "credit_hours":3, "demand":[ {"Prof":"Wechsler", "No_of_Students":14} ] },
{"courses":"CS685"  , "credit_hours":3, "demand":[ {"Prof":"Dana"    , "No_of_Students":32},
                                                   {"Prof":"Tecuci"  , "No_of_Students":27} ] },
{"courses":"CS773"  , "credit_hours":3, "demand":[ {"Prof":"Dana"    , "No_of_Students":38},
                                                   {"Prof":"Wechsler", "No_of_Students":37} ] },
{"courses":"CS787"  , "credit_hours":3, "demand":[ {"Prof":"Brodsky" , "No_of_Students":27} ] },
{"courses":"CS795"  , "credit_hours":3, "demand":[ {"Prof":"Dana"    , "No_of_Students":36} ] },
{"courses":"SWE645" , "credit_hours":3, "demand":[ {"Prof":"Wechsler", "No_of_Students":15} ] },
{"courses":"INFS740", "credit_hours":3, "demand":[ {"Prof":"Brodsky" , "No_of_Students":29} ] });

```

OBJECTIVE

The objective of the recommender system is to assign each course a professor who has the expertise to teach the course such that the Average Professor Satisfaction Ratio is maximized given the following constraints and then evaluate the assignment on a number of metrics.

CONSTRAINTS

The optimization of objective, the Average Professor Satisfaction Ratio (being maximized) was subject to the following constraints:

- Only one professor can be assigned to each course. This constraint can be achieved by specifying that sum of flags (the decision variables) for a particular course being offered in a semester should be 1.
- An additional constraint was used specifying the values of flags can be either 0 or 1.
- Each professor must be allotted a number of courses to teach for every semester that lies between the minimum and the maximum number of courses he can teach for the given semester.
- Standard deviation range must be specified for which the Average Professor Satisfaction Ratio is being computed. Given the range, optimal solution will be found for that range of standard deviation

```
(:Constraints.....................................:)

let $flagSumCons := for $c in $flagcondition
                    let $innerconst := every $vc in $c.Value[] satisfies $vc.flag = 1
                    return $innerconst

let $flagSumConstraint := every $i in $flagSumCons satisfies $i = true

let $profflags := for $pf in $input.profcoursepairs[]
                  let $flags := for $p in $input.professors[].pname
                                let $flagcount := fn:sum(for $fg in $pf.List[]
                                                         where $fg.Prof = $p
                                                         return $fg.flag)
                                return {Prof:$p ,fcount:$flagcount}
                  return{sem:$pf.sem,Flags:$flags}

let $minMaxConstr := for $pfg in $profflags
                     let $ppref := for $pr in $input.prof_prefer[]
                                   where $pfg.sem = $pr.sem
                                   let $prefcount := for $pcount in $pfg.Flags[]
                                                     let  $min := $pr.($pcount.Prof).min
                                                     let $max  :=  $pr.($pcount.Prof).max
                                                     let $check := $min le $pcount.fcount and $pcount.fcount le $max
                                                     return $check
                                   return $prefcount
                     let $innerconst := every $vc in $ppref satisfies $vc = true
                     return $innerconst

let $minMaxConstraint := every $i in $minMaxConstr satisfies $i = true

let $binaryFlagConstraint := every $x in $input.profcoursepairs[].List[].flag satisfies ($x ge 0 and $x le 1)

let $constraints := (($flagSumConstraint and $minMaxConstraint) and $binaryFlagConstraint)
                      and (($standardDeviation le $input.maxSD and $standardDeviation ge $input.minSD))
```

METRICS

The following metrics are involved in the performance evaluation of the recommender system:

- Professor Satisfaction Ratio - it gives a measure of how satisfied a professor would be on account of satisfaction with respect to the preferences of courses he wanted
- Average Professor Satisfaction Ratio - a measure based on the average of professors' satisfaction (allotment of their preferred courses). It is a normalized value between 0 and 1.
- Overall Unfairness of the Allotment - a measure determining how unfair or biased the allotment is towards taking into account the preferences of all professors. It is computed as the standard deviation of the Professor Satisfaction Ratio from the Average Ratio.
- Core Courses Offered Per Semester - gives the number of core courses offered every semester and their Course IDs.
- Full Time Equivalents – a measure of the average number of full time equivalents every semester. It is computed as a product of Credit Hours of a course * the number of students taking the course

```
(: Metrics.............................................. :)

(: Satisfaction of each professor per sem  :)
let $satispersem := for $s in $input.courses[]
                    let $sems := for $sps in $profsatis
                                 let $inside := for $i in $sps.satisfacPerSem[]
                                                where $i.sem = $s.sem
                                                return $i.satisfac
                                 return $inside
                    return {sem:$s.sem, Satisfaction:$sems}


(:mean satisfaction per semester  :)
let $meansemsatis := for $m in $satispersem
                     return {sem : $m.sem, mean :fn:sum($m.Satisfaction[]) div 4}


(:minimum and maximum satisfaction for each semester:)
let $minmax := for $m in $satispersem
               let $maxi := fn:max((for $mx in $m.Satisfaction[] return $mx))
               let $mini := fn:min((for $mn in $m.Satisfaction[] return $mn))
               let $diff := $maxi - $mini
               return {sem:$m.sem, ratio: $diff}


(:Standard deviation per sem :)
let $sdpersem := for $sps in $satispersem
                 let $sd:= for $ms in $meansemsatis
                           where $sps. sem = $ms.sem
                           let $compute:= fn:sum (for $s in $sps.Satisfaction[]
                                                  return( fn:abs($s - $ms.mean ) ) ) div 4
                           return $compute

                 return {sem: $sps.sem, SD:$sd}


let $coreCourses := for $sems in $input.courses[]
                    let $courses := for $coursesoff in $sems.courses[]
                                    return if ($coursesoff eq $input.core_courses[] [$$ eq $coursesoff])
                                           then $coursesoff
                                           else ()
                    return {sem:$sems.sem,Count: count($courses),Courses: $courses}
```

```
  let $ftePerSem := for $sems in $input.courses[]
                    let $outside := fn:sum(for $profs in $profcourses
                                           let $inside := for $insems in $profcourses.sems[]
                                                          where $insems.sem = $sems.sem
                                                          let $allcourses := for $course in $insems.courses[]
                                                                             let $insidecourse := for $cd in $input.course_demand[],$dem in $cd.demand[]
                                                                                 where ($cd.courses = $course.course and $dem.Prof = $profs.Prof)
                                                                                 return fn:sum($cd.credit_hours * $dem.No_of_Students * $course.flag) div 12

                                                                             return $insidecourse
                                                          return $allcourses
                                           return $inside)
                    return {sem:$sems.sem, fte:$outside}

  let $utility := $avgSatis -  $standardDeviation

  return {AvgProfSatisfaction: $avgSatis, constraints: $constraints,  CoreCourses: $coreCourses, FTEs: $ftePerSem, Unfairness: $standardDeviation,
  Utility: $utility}

};
```

DGAL

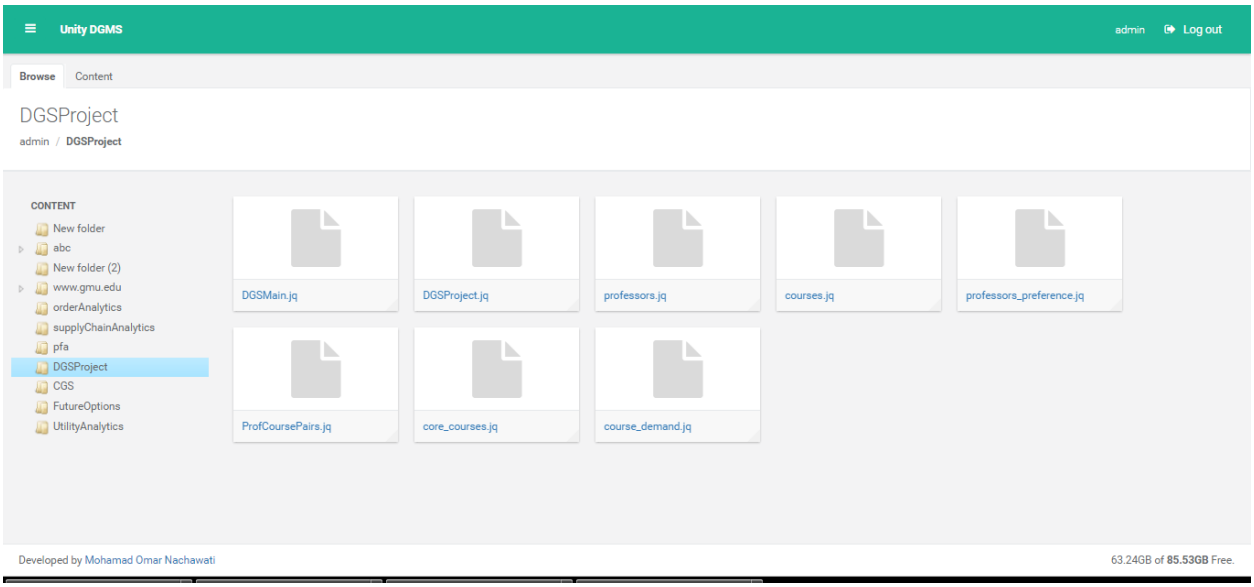
DGAL helps in finding the optimal results by either maximizing or minimizing the value of the decision variable that has been chosen.

We incorporated DGAL into the project to find the optimal Professor and Course allotment by setting the flag variable to o or

Steps involved in Using DGAL :

- Connect to the server at unity.vsnet.gmu.edu using OpenVPN.
- Created a Folder " DGSProject" which contains DGSProject.jq, the analytical function to compute the allotment, the constraints and the metrics.
- DGSMain.jq, the file used to run the Analytics function.
- Our goal was to maximize the average Professor satisfaction and thus we use the argmax() function .

The project can be found under the following URL : https://unity.vsnet.gmu.edu/admin/DGSProject



DGSMain.jq

```
jsoniq version "1.0";


import module namespace dgal ="http://mason.gmu.edu/~mnachawa/dgal.jq";

import module namespace ns  = "http://DGSProject/DGSProject.jq";
import module namespace ns1 = "http://DGSProject/professors.jq";
import module namespace ns2 = "http://DGSProject/courses.jq";
import module namespace ns3 = "http://DGSProject/professors_preference.jq";
import module namespace ns4 = "http://DGSProject/core_courses.jq";
import module namespace ns5 = "http://DGSProject/ProfCoursePairs.jq";
import module namespace ns6 = "http://DGSProject/course_demand.jq";


let $input :=( { professors:$ns1:professors, courses:$ns2:courses, prof_prefer:$ns3:professors_preference,
                 core_courses:$ns4:core_courses, profcoursepairs:$ns5:ProfCoursePairs, course_demand:$ns6:course_demand } )



return (

for $i in (0.060, 0.075, 0.090, 0.105, 0.120, 0.135, 0.150, 0.165, 0.180, 0.195)

return ns:DGSProject(dgal:argmax({|$input,{minSD: $i, maxSD: $i + 0.015}|}, ns:DGSProject#1,"AvgProfSatisfaction",{language: "opl", solver : "cplex"}))

 )

```

DGSProject.jq

```
jsoniq version "1.0";

module namespace ns = "http://DGSProject/DGSProject.jq";
import module namespace math = "http://www.w3.org/2005/xpath-functions/math";
import module namespace xs = "http://www.w3.org/2005/xpath-functions";
declare function ns:DGSProject($input)
  {

    let $flagcondition :=  for $cff in $input.courses[], $i in $input.profcoursepairs[]
                           where $i.sem = $cff.sem
                           let $innerconst := for
                           $ic in $cff.courses[]
                                        let $inside := fn:sum( for $li in $i.List[]
                                                               where $ic = $li.course
                                                               return $li.flag)
                                        return {course:$ic, flag:$inside}
                           return  {sem: $cff.sem, Value:$innerconst}


    let $profsatisfaction := for $profsat in $input.profcoursepairs[]
                             let $semsatis := for $ss in $profsat.List[]
                                        return{course:$ss.course, Prof: $ss.Prof, flag :$ss.flag}
                             return {sem:$profsat.sem,Courses_Offered:$semsatis}


    let $profcourses := for $p in $input.professors[].pname
                        let $c := for $sems in $profsatisfaction
                                  let $prof := for $course in $sems.Courses_Offered[]
                                               where $course.Prof=$p
                                               return {course:$course.course,flag :$course.flag}
                                  return {sem:$sems.sem,courses:$prof}
                        return {Prof:$p, sems:$c}


    let $preferences := for $p in $input.professors[].pname
                        let $prefs := for $sems in $input.prof_prefer[]
                                      return {sem:$sems.sem,preferences:[$sems.($p).pref1,$sems.($p).pref2]}
                        return {Prof:$p,sems:[$prefs]}

```

OUTPUT


```
    ],
  - profcoursepairs: [
      - {
            sem: 1,
          - List: [
              - {
                    course: "CS550",
                    Prof: "Brodsky",
                    flag: 1
                },
              - {
                    course: "INFS740",
                    Prof: "Brodsky",
                    flag: 1
                },
              - {
                    course: "CS681",
                    Prof: "Tecuci",
                    flag: 1
                },
              - {
                    course: "CS685",
                    Prof: "Tecuci",
                    flag: 1
                },
              - {
                    course: "CS583",
                    Prof: "Tecuci",
                    flag: 0
                },
```

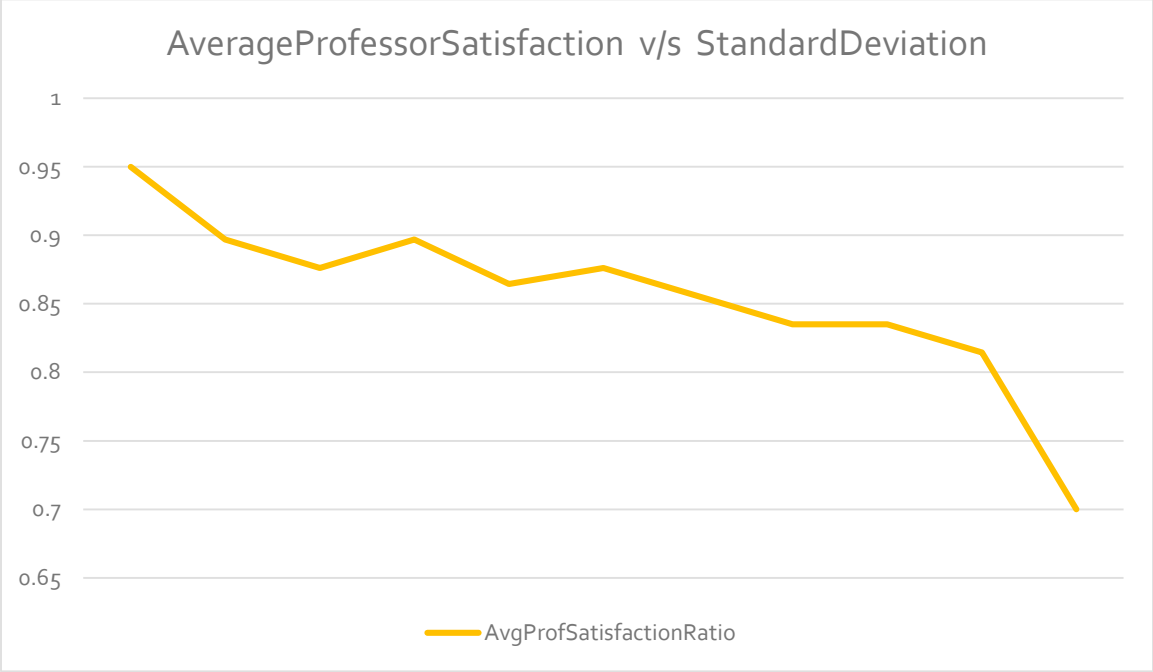
```
                {
                    course: "CS795",
                    Prof: "Dana",
                    flag: 1
                }
            ]
        },
      - {
            sem: 2,
          - List: [
              - {
                    course: "CS550",
                    Prof: "Brodsky",
                    flag: 1
                },
              - {
                    course: "CS650",
                    Prof: "Brodsky",
                    flag: 1
                },
              - {
                    course: "CS583",
                    Prof: "Tecuci",
                    flag: 0
                },
              - {
                    course: "CS685",
                    Prof: "Tecuci",
                    flag: 0
                },
              - {
                    course: "CS580",
```

{ "AvgProfSatisfaction" : 0.896875, "constraints" : true, "Unfairness" : 0.061875, "Utility" : 0.835 }{ "AvgProfSatisfaction" : 0.87625, "constraints" : true, "Unfairness" : 0.0825, "Utility" : 0.79375 }{ "AvgProfSatisfaction" : 0.896875, "constraints" : true, "Unfairness" : 0.103125, "Utility" : 0.79375 }{ "AvgProfSatisfaction" : 0.814375, "constraints" : true, "Unfairness" : 0.1134375, "Utility" : 0.7009375 }{ "AvgProfSatisfaction" : 0.87625, "constraints" : true, "Unfairness" : 0.12375, "Utility" : 0.7525 }{ "AvgProfSatisfaction" : 0.855625, "constraints" : true, "Unfairness" : 0.144375, "Utility" : 0.71125 }{ "AvgProfSatisfaction" : 0.835, "constraints" : true, "Unfairness" : 0.165, "Utility" : 0.67 }{ "AvgProfSatisfaction" : 0.835, "constraints" : true, "Unfairness" : 0.165, "Utility" : 0.67 }{ "AvgProfSatisfaction" : 0.814375, "constraints" : true, "Unfairness" : 0.185625, "Utility" : 0.62875 }{ "AvgProfSatisfaction" : 0.79375, "constraints" : true, "Unfairness" : 0.20625, "Utility" : 0.5875 }

The images above represent the various outputs of DGAL. The outputs obtained have optimized values.

DGAL uses the cplex solver and converts JSONiq to OPL to solve the optimization problem. The metrics have been evaluated on optimized output of flags which are the decision variables.

GRAPHICAL REPRESENTATION OF OUTPUT



The output of DGAL for determining the values of Average Professor Satisfaction Ratio for a standard deviation in a specific range is shown above.

As we can see from the graph, the standard deviation increases as the Average Satisfaction Ration decreases. There is an inverse relation between the two. Intuitively, If the Average Satisfaction Ratio is 1, i.e., the maximum possible, the corresponding standard deviation or unfairness measure will be 0, hence the inverse relationship.

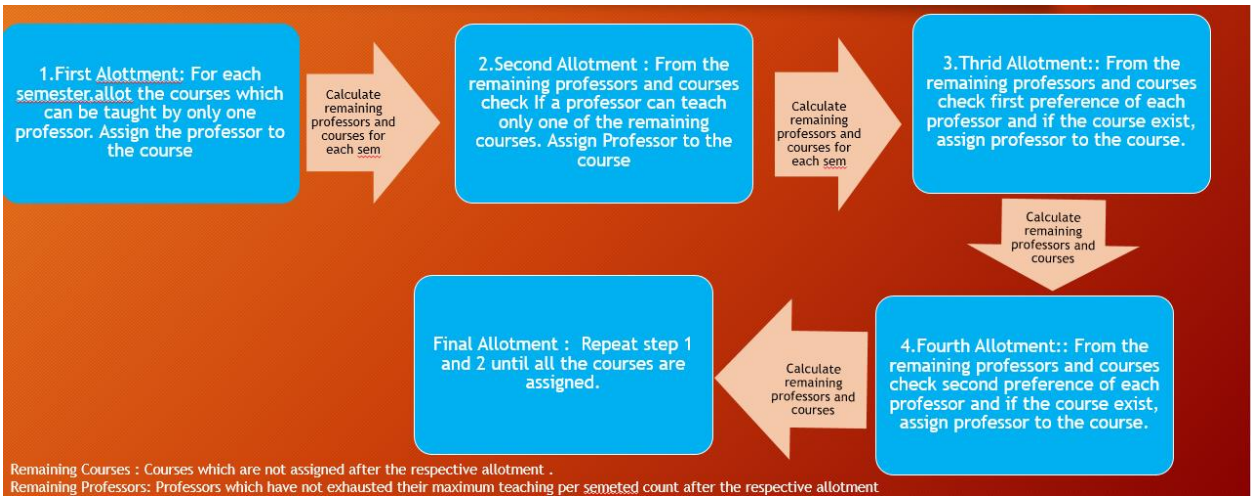
The Average Satisfaction Ratio values were generated by maximizing Average Satisfaction Ratio on bounding standard deviation between a minimum and maximum value. We receive values for standard deviation in the max-min interval that gives the maximum possible Average Satisfaction.

CONCLUSION

- We tried manually optimizing a way to allot courses by defining a flow algorithm which does take a lot of possibilities of the constraints, yet, misses important failure cases. The process of manual automation was successful to a certain extent, but does not backtrack to check for the best possible solution.
- With the help of DGAL, we were able to build the proposed recommender system that optimizes the allotment and gives the most optimal answer given the input, the objective and the constraints.
- The DGAL optimizer gives a possible allotment of courses in the form of values of flags associated with courses, professors and then the metrics can be evaluated on the optimal allotment
- This recommender system can help automated allotment of courses in universities thereby reducing human effort.

APPENDIX: MANUAL OPTIMIZATION VS DGAL OPTIMIZATION

We also tried to solve this problem by finding an algorithm which will automate the process. The flow of our algorithm is as follows



However there were many problems with this process:

1) It did not always provided the optimized result
2) It was not scalable and robust
3) Backtracking was not done, so there were chances of getting wrong output.

The optimization with DGAL improved the shortcomings of the manual optimization algorithm and made our solution more robust. It returned the optimized results with given constraints and objective of maximizing professor satisfaction ratio and overall fairness.

We have added screenshots from the manual implementation done in JSONiq below for reference.

```
jsoniq version "1.0";

module namespace ns = "DGSProject";

declare function ns:projectAnalytics($input)
{
(:---- list of all the courses (per sem) along with the professors who can teach the course  (per course structure)------:)
    let $CourseTeachable := (for $co in $input.courses[]
      let $course:= for $c in $co.courses[]
                    let $assign:= for $p in $input.professors[]
                                  let $w := for $pi in $p.can_teach[] where $c = $pi return $p.pname
                                  return $w
                    return {course: $c, Prof : $assign , No_of_Prof: count($assign)}
      return {sem:$co.sem,Courses_Offered:$course})

(:--------- list of courses (per sem) that a professor can teach (per professor structure) ----------:)

 let $prof_can_teach_sem:=for $sems in $input.courses[]
                          let $prof_teach:= for $p in $input.professors[]
                                            let $courseoff := for $c in $sems
                                                let $courses := for $course in $c.courses[]
                                                      let $match :=  for $pi in $p.can_teach[] where $course = $pi
                                                      return $course
                                                return $match
                                            return $courses
                                   return {Proff_id:$p.pid,Proff_Name:$p.pname,Can_Teach_Courses:$courseoff}

                            return {sem:$sems.sem,List:$prof_teach}
```

```
  let $onecourse:=  for $y in $CourseTeachable
                    let $courses:=for $x in $y.Courses_Offered[] where $x.No_of_Prof=1

                    return {course: $x.course, Prof : $x.Prof}
      return {sem:$y.sem,Courses_Offered:$courses}

    let $remcourseAfterone := ns:remainingcourses ({remcourse:$CourseTeachable,allotment:$onecourse})
    let $remProffessorsAfterone := ns:remainingProfessors ({concatenated:$onecourse,inputProf:$input.professors[],inputPref: $input.proff_


  let $intermediateAllotmentfrst := ns:intermediateAllotment({remainProf :$remProffessorsAfterone, remaincourses : $remcourseAfterone})
  let $concatenate :=   ns:concatenateResults ({one:$onecourse,two:$intermediateAllotmentfrst})

  let $remProffessorsafterInter := ns:remainingProfessors ({concatenated:$concatenate,inputProf:$input.professors[],inputPref: $input.prof
  let $remcourseafterintermediate := ns:remainingcourses ({remcourse:$remcourseAfterone,allotment:$intermediateAllotmentfrst})

  let $frstprefalott := ns:assignbypref({inputPref: $input.proff_prefer[],remainingcourses:$remcourseafterintermediate,remainingProfessors

  let $distinct_course := distinct-values(for $temp in $frstprefalott.Courses_Offered[]
                                          return $temp.course)

    let $countCalculation:= for $tsa in $frstprefalott
                            let $discount := for $dc in $distinct_course
                                             let $compare:= for $cmp in $tsa.Courses_Offered[]
                                                            where $dc = $cmp.course
                                                            return count($cmp.course)

                                             return {course:$dc, Count: count($compare)}
```

```
let $countCalculation:= for $tsa in $frstprefalott
                        let $discount := for $dc in $distinct_course
                                        let $compare:= for $cmp in $tsa.Courses_Offered[]
                                                      where $dc = $cmp.course
                                                      return count($cmp.course)

                                        return {course:$dc, Count: count($compare)}

                        return {sem:$tsa.sem,Courses_Offered:$discount}


let $actualPref1 := for $tsa in $frstprefalott
                    let $countCal := for $cC in $countCalculation
                                    where $tsa.sem = $cC.sem
                                    let $inside := for $t in $tsa.Courses_Offered[], $c in $cC.Courses_Offered[]
                                                  where $t.course = $c.course and $c.Count =1
                                                  return  {course:$t.course, Prof:$t.Prof}

                                     return $inside
                    return {sem:$tsa.sem,Courses_Offered:[$countCal]}

let $concatenateactualPref1 :=   ns:concatenateResults ({one:$concatenate,two:$actualPref1})
let $remProffessorsAfteractualPref1 := ns:remainingProfessors ({concatenated:$concatenateactualPref1,
inputProf:$input.professors[],inputPref: $input.proff_prefer[]})
let $remcourseactualPref1 := ns:remainingcourses ({remcourse:$remcourseafterintermediate,allotment:$actualPref1})


let $intermediateAllotmentsecond := ns:intermediateAllotment({remainProf :$remProffessorsAfteractualPref1, remaincourses : $remcourseactualPref1})
```

```
    (: Constraints ............................. :)

    (: Inner constraint1 calculation :)
    let $innerconstraint := for $cff in  $concatenatefinal.Courses_Offered[]
                            let $innerconst := for $i in $input.professors[]
                                               where $i.pname = $cff.Prof
                                               return if (($i.can_teach[])[contains($$,$cff.course)])
                                               then true
                                               else false
                            return  $innerconst

    let $constraint1 := every $i in $innerconstraint satisfies $i = true

    (: Inner constraint2 calculation :)
    let $constr2:=for $p in $input.professors[].pname
                  let $pr:=for $prof in $input.proff_prefer[]
                           let $check:=for $sems in $concatenatefinal
                                       where $prof.sem = $sems.sem
                                       return ($prof.($p).min <= sum(for $x in $sems.Courses_Offered[] where $x.Prof=$p return 1) and sum(for $x in $se
                           return $check
                  return $pr

    let $constraint2:=every $x in $constr2 satisfies true

    let $allconstraints := $constraint1 and $constraint2


    (: Core courses ..................  :)
    let $coreCourses := for $cf in $concatenatefinal
```

```
(:---------- average satisfaction ratio of each professors per semester ----------------:)

let $profsatis:=for $off in $profcourses
                let $p:=for $pref in $preferences
                        where $pref.Prof=$off.Prof
                        let $po:=for $x in $off.sems[]
                                 let $pr:=for $y in $pref.sems[]
                                          where $y.sem=$x.sem
                                          let $pen:=for $c in $x.courses[]
                                                    return if(($y.preferences[])[contains($$,$c)])
                                                           then (0)
                                                           else (-0.4)
                                          return (if((1+sum($pen)) > 0) then (1+sum($pen)) else 0)
                                 return {sem:$x.sem, satisfac:$pr}
                        return $po
                return {Prof:$off.Prof,satisfacPerSem:$p}

(:---------- average satisfaction ratio of each professor (among all semesters) ----------------:)

let $profAvgSatis:=for $p in $profsatis
                   let $s:=sum(for $sem in $p.satisfacPerSem[] return $sem.satisfac) div 4
                   return {Prof:$p.Prof, ProfAvgSatisfac: $s}

(:----------average satisfaction ratio among all professors----------------:)

let $avgSatis:={AvgSatisRatio: sum(for $p in $profAvgSatis return $p.ProfAvgSatisfac) div sum(for $p in $profAvgSatis return 1)}

(:-- jsoniq does not hav square root function so cant calculate standard devation, hence varience for fairness measure --:)
```

```
(:----------average satisfaction ratio among all professors----------------:)

let $avgSatis:={AvgSatisRatio: sum(for $p in $profAvgSatis return $p.ProfAvgSatisfac) div sum(for $p in $profAvgSatis return 1)}

(:-- jsoniq does not hav square root function so cant calculate standard devation, hence varience for fairness measure --:)

let $fairnessVarience:=sum(for $pa in $profAvgSatis
                           return ($pa.ProfAvgSatisfac - $avgSatis.AvgSatisRatio)*($pa.ProfAvgSatisfac - $avgSatis.AvgSatisRatio))
                           div sum(for $p in $profAvgSatis return 1)

(:-- FTE or Full_Time_Equivalent is calculated by credit_hours_of_course * number_of_students_taking_course / 12 --:)
(:----------------fte per sem-------------:)

let $ftePerSem:=for $x in $concatenatefinal
                let $persem:=sum(for $y in $x.Courses_Offered[]
                                 let $courses:=for $cd in $input.course_demand[],$dem in $cd.demand[]
                                               where $cd.course = $y.course and $dem.Prof = $y.Prof
                                               return $cd.credit_hours * $dem.No_of_Students div 12
                                 return $courses)
                return {sem:$x.sem, fte:$persem}

(:----------------total fte over 4 sems--------------:)
let $fteTotal:=sum(for $s in $ftePerSem return $s.fte) div 4


(:return {FinalAllotment:$concatenatefinal,ConstraintsSatisfied:$allconstraints,CoreCourses:$coreCourses,AverageProfessorSatisfactionRatio:$avgSatis,fairne

:)
return $ftePerSem
```

```
(:--------------------------- other functions called in code---------------------------:)
declare function ns:remainingcourses($remcourse_and_allotment)
{
let $remafter :=  for $rem in $remcourse_and_allotment.remcourse[], $a in $remcourse_and_allotment.allotment[]
                  where $rem.sem  eq $a.sem
  let $remain := for $main in $rem.Courses_Offered[]
                    return  $main.course
      let $alloted:= distinct-values(for $al in $a.Courses_Offered[]
                      return $al.course)

let $final := for $r in $remain
              return if(($alloted)[contains($$,$r)])
                then ()
               else $r

let $finalstruct := for $fin in $final
let   $structure    := for $struct in $rem.Courses_Offered[]
                        where $struct.course = $fin
                       return {course: $struct.course, Prof : $struct.Prof}
                       return $structure


               return {sem:$rem.sem,Courses_Offered:[$finalstruct]}


               return $remafter

       };
```

```
 declare function ns:intermediateAllotment($remains)
 {
   let $remainp :=  for $rp in $remains.remainProf[]
   let $remp := for $main in $rp.RemCount[]
                return  $main.Prof
              return {sem:$rp.sem, remp: [$remp]}


   (:)-- no. of courses a professor can teach from the remaining courses ---------------:)

   let $noProfCanTeachFromRem:= for $sems in  $remains.remaincourses[]
                                let $profs := for $rem in $remainp where $rem.sem = $sems.sem return $rem.remp[]
                                let $profcount:=for $p in $profs
                                               let $count:=for $x in $sems.Courses_Offered[].Prof[]
                                                           where $x=$p
                                                           return count($x)
                                                order by sum($count)
                                                return {$p : sum($count)}
                                return {sem:$sems.sem,can_teach:$profcount}

   (:)----------- if a prof can teach only 1 course from remaining courses, assign it to him -------------:)

   let $intermediateAllotment:=for $sem in $remains.remaincourses[]
                               let $courses:=for $x in $sem.Courses_Offered[]
                                             let $y:= for $profcount in $noProfCanTeachFromRem
                                                      where $profcount.sem = $sem.sem
                                                      let $profallot:= for $z in $x.Prof[]
                                                                       where $profcount.can_teach[].($z) =1
                                                                       return {course:$x.course,Prof:$z}
```

The output obtained on running the JSONiq module was as follows:


```
{ "FinalAllotment" : [ { "sem" : 1, "Courses_Offered" : [ { "course" : "CS550", "Prof" : "Brodsky" }, { "course" : "INFS740", "Prof" : "Brodsky" }, { "course" : "CS681"
, "Prof" : "Tecuci" }, { "course" : "SWE645", "Prof" : "Wechsler" }, { "course" : "CS795", "Prof" : "Dana" }, { "course" : "CS773", "Prof" : "Wechsler" }, { "course" :
"CS685", "Prof" : "Tecuci" }, { "course" : "CS583", "Prof" : "Dana" } ] }, { "sem" : 2, "Courses_Offered" : [ { "course" : "SWE645", "Prof" : "Wechsler" }, { "course" :
 "CS550", "Prof" : "Brodsky" }, { "course" : "CS650", "Prof" : "Brodsky" }, { "course" : "CS681", "Prof" : "Tecuci" }, { "course" : "CS580", "Prof" : "Tecuci" }, { "cou
rse" : "CS773", "Prof" : "Wechsler" }, { "course" : "CS685", "Prof" : "Dana" }, { "course" : "CS583", "Prof" : "Dana" } ] }, { "sem" : 3, "Courses_Offered" : [ { "cours
e" : "CS650", "Prof" : "Brodsky" }, { "course" : "CS787", "Prof" : "Brodsky" }, { "course" : "CS681", "Prof" : "Tecuci" }, { "course" : "CS682", "Prof" : "Wechsler" },
{ "course" : "SWE645", "Prof" : "Wechsler" }, { "course" : "CS580", "Prof" : "Tecuci" }, { "course" : "CS583", "Prof" : "Dana" }, { "course" : "CS685", "Prof" : "Dana"
} ] }, { "sem" : 4, "Courses_Offered" : [ { "course" : "INFS740", "Prof" : "Brodsky" }, { "course" : "CS550", "Prof" : "Brodsky" }, { "course" : "CS682", "Prof" : "Wech
sler" }, { "course" : "CS795", "Prof" : "Dana" }, { "course" : "CS580", "Prof" : "Tecuci" }, { "course" : "CS583", "Prof" : "Dana" }, { "course" : "CS685", "Prof" : "Te
cuci" }, { "course" : "CS773", "Prof" : "Wechsler" } ] } ], "ConstraintsSatisfied" : true, "CoreCourses" : [ { "sem" : 1, "Core_Courses" : { "Count" : 2, "Courses" : [
"CS550", "CS583" ] } }, { "sem" : 2, "Core_Courses" : { "Count" : 3, "Courses" : [ "CS550", "CS580", "CS583" ] } }, { "sem" : 3, "Core_Courses" : { "Count" : 2, "Course
s" : [ "CS580", "CS583" ] } }, { "sem" : 4, "Core_Courses" : { "Count" : 3, "Courses" : [ "CS550", "CS580", "CS583" ] } } ], "AverageProfessorSatisfactionRatio" : { "Av
gSatisRatio" : 0.875 }, "fairnessVarience" : 0.006875, "TotalFTEs" : 63.875 }
```


Project Contributions:

All team members have contributed equally towards implementing the Recommender System and have always been together while discussing the implementation or while coding the project.

Acknowledgements:

We would like to thank Omar for all his help with DGAL.