Optimize Your 32 - Course Selection as a CLP Problem

December 9, 2019

Matt Bock and Blake Bullwinkel

Note: This notebook uses OR-Tools for Python. All other dependencies are available as part of the standard Anaconda installation.

1 Introduction

In this project, we treat the selection of one's 32 courses at Williams as an optimization problem using linear programming. We implement constraints to ensure that the selection of courses is valid; for example, it must be free of scheduling conflicts, and courses must have the proper prerequisites taken in order to be selected. We then optimize the course selection (given a few input parameters such as the intended major(s) of the student) with an objective function derived from student reviews of the courses (from Factrak). We use a subset of the course catalog as our data set. This includes the full curriculum for six subjects: 1. ARTH 2. CSCI 3. ECON 4. ENGL 5. MATH 6. STAT

We have data for all of these courses from the past four semesters (Fall 2018 through Spring 2020). We make the simplifying assumption that all courses are offered on four-semester cycles - that is, if a given course is offered Fall 2018, then it will be offered in Fall 2020.

1.1 Project Objectives

- Apply linear programming concepts learned in the course to an actual solver with working code
- Combine data from multiple sources in order to create a tool that provides insight into course seection at Williams

2 Course Scheduling as a CLP

Before looking at our implementation, we explain how course scheduling can be formulated as a linear programming problem.

2.1 Parameters

- Class number $n \in \{1, ..., N\}$
- Prefix $p \in \{1, ..., P\}$
- $n_p = \{\text{the number of courses with prefix } p\}$
- Division $d \in \{1, 2, 3\}$

- Semester $s \in \{1, ..., 8\}$
- Time slot $t \in \{1, ..., T\}$
- Lists of courses D, Q and W, indicating courses that meet DPE, QFR and Writing Intensive requirements, respectively.

2.2 Variables

Our solution consists of boolean indicator variables of this form: - $x_{ndpst} = \{1$ if taking course n in division d with prefix p during semester s at time t, 0 otherwise$\}$

We have lists of which courses meet certain requirements, such as those that meet DPE, QFR and Writing Intensive requirements. We can model these lists as variables with respect to our defined parameters; these variables will form some of the constraints applied to our solution variables.

- $y_p = \{1$ if have taken at least one course with prefix p, 0 otherwise$\}$
- $w_{ns} = \{1$ if course n taken during semester s is writing intensive, 0 otherwise$\}$
- $d_{ns} = \{1$ if course n taken during semester s is DPE, 0 otherwise$\}$
- $q_{ns} = \{1$ if course n taken during semester s is QFR, 0 otherwise$\}$
- $a_s = \{1$ if studying abroad during semester s, 0 otherwise$\}$

2.3 Constraints

2.3.1 32 Credits

In order to graduate, the student must pass 32 semester classes (this term will be summed over all valid $ndpt$ combinations for the given s, since the code only creates those variables):

$$\sum_{s=1}^{8} x_{ndpst} = 32$$

2.3.2 General Distribution Requirements

Divisional: By the time of graduation, the student must have taken at least three courses in each division:

$$\text{for each } d \in \{1, 2, 3\} : (\sum_{s=1}^{8} x_{ndpst} > 2)$$

In each division, the student must take courses with at least two different prefixes:

for each $d \in \{1, 2, 3\}$:

$$\text{for each } p \in \{1, ..., P\} : y_p \leq \sum_{n=1}^{N} x_{ndpst}, y_p \geq \frac{\sum_{n=1}^{N} x_{ndpst}}{n_p}$$

$$\sum_{p=1}^{P} y_p > 1$$

Writing Intensive: By the end of sophomore year, the student must have taken at least one writing intensive course:

$$\sum_{s=1}^{4} w_{ns} > 0$$

By the end of junior year, the student must have taken at least two writing intensive courses:

$$\sum_{s=1}^{6} w_{ns} > 1$$

DPE: By the end of senior year, the student must have taken at least one DPE course:

$$\sum_{s=1}^{8} d_{ns} > 0$$

QFR: By the end of junior year, the student must have taken at least one QFR course:

$$\sum_{s=1}^{6} q_{ns} > 0$$

2.3.3 Underclassmen Distribution Requirements

Freshmen: In freshman year, the student cannot take two or more courses with the same prefix per semester:

$$\text{for each } s \in \{1,2\} : \text{for each } p \in \{1,...,P\} : \sum_{n=1}^{N} x_{ndpst} < 2$$

Sophomores: In sophomore year, the student cannot take three or more courses with the same prefix per semester:

$$\text{for each } s \in \{3,4\} : \text{for each } p \in \{1,...,P\} : \sum_{n=1}^{N} x_{ndpst} < 3$$

In sophomore year, the student cannot take four or more courses with the same prefix:

$$\text{for each } p \in \{1,...,P\} : \sum_{s=3}^{4} x_{ndpst} < 4$$

First two years: Over the course of freshman and sophomore year, the student cannot take six or more courses with the same prefix:

$$\text{for each } p \in \{1,...,P\} : \sum_{s=1}^{4} x_{ndpst} < 6$$

By the end of sophomore year, the student must have taken at least two courses in each division:

$$\text{for each } d \in \{1,2,3\} : (\sum_{s=1}^{4} x_{ndpst} > 1)$$

2.3.4 Study Abroad

If the student is studying abroad for all of junior year, then writing intensive and QFR requirements must be completed by the end of sophomore year:

$$\text{IF } (\sum_{s=5}^{6} a_s = 2) \text{ THEN } (\sum_{s=1}^{4} w_{ns} > 1 \text{ AND } \sum_{s=1}^{6} q_{ns} > 0)$$

If the student is studying abroad for only one semester, then all requirements are covered by general distribution requirements.

2.3.5 Logistical

In each semester, no two courses can be taken at the same time:

$$\text{for each } s \in \{1, ..., 8\} : \text{for each } t \in \{1, ..., T\} : \sum_{n=1}^{N} x_{ndpst} < 2$$

[9]:
```
import json
from ortools.linear_solver import pywraplp
import numpy as np
from datetime import time
import itertools
import copy
```

3 Data Sources

This project uses three data sets from two sources. We have two data sets representing the course catalog - one from WSO and one form the registrar's office. Both data sets include all basic course information (prefix, catalog number, meeting time, etc.) for the relevant terms. However, each also gives us some unique information. Notably, the registrar's data set gives us the cross-listing relations between different desigations of the same course, while the WSO data gives us the distribution requirements met by each course.

The third data set that we use is Factrak data. We join this data set with the other two in order to associate student recommendations of courses with the catalog data.

3.1 Parsing in Data

Here we define a 'Course' Data structure that contains both the catalog data for a course and a reference to its indicator variable in the optimization library.

[10]:
```
# The Course class will be our data structure for storing data associated with␣
 ↪each course
class Course:
    def __init__(self, params):
        self.data = copy.deepcopy(params)

    def __eq__(self, other):
        return self.data["optimizer_id"] == other.data["optimizer_id"]

    def __ne__(self, other):
        return self.data["optimizer_id"] != other.data["optimizer_id"]

    def __lt__(self, other):
        return self.data["optimizer_id"] < other.data["optimizer_id"]

    def __le__(self, other):
        return self.data["optimizer_id"] <= other.data["optimizer_id"]
```

```
    def __gt__(self, other):
        return self.data["optimizer_id"] > other.data["optimizer_id"]

    def __ge__(self, other):
        return self.data["optimizer_id"] >= other.data["optimizer_id"]
```

[12]:
```
# Parse 19-20 data
with open("data_joined_unofficial.json") as json_file:
    data = json.load(json_file)

courses = []
terms = ["1191", "1193", "1201", "1203", "1211", "1213", "1221", "1223"]
for i in range(8):
    courses.append([])
    c_list = courses[i]
    for d in data:
        if terms.index(d["Term"]) == i:
            c = Course(d)
            c.data["Semester"] = i
            c_list.append(c)
        elif terms.index(d["Term"]) == (i % 4):
            c = Course(d)
            t = int(c.data["Term"])
            t += 20
            c.data["Term"] = str(t)
            c.data["STRM"] = str(t)
            c.data["Semester"] = i
            c.data["optimizer_id"] += len(data)
            c_list.append(c)

# with open("factrak_survey_data.json") as json_file:
#     factrak_survey_data = json.load(json_file)
for lst in courses:
    lst.sort()
```

4 Implementation using OR-Tools

This project uses Google's or-tools library for python. We create a mixed integer solver using or-tools, and create a boolean indicator variable for each course as explained above. Note that, since we create a variable for each actual course in the data set, we only create a variable x_{ndpst} if there is actually a course that exists for that combination of parameters. Additionally, since we are able to use data structures in this way, some of the linear constraints may deviate slightly from the math presented above. For example, we do not require divisional requirements to be fullfilled by two different subjects because this would be impossible with our data set, as ECON is the only Division 2 subject that we have.

[13]:
```
solver = pywraplp.Solver("course_scheduling_program", pywraplp.Solver.
 ↪CBC_MIXED_INTEGER_PROGRAMMING)

# create variables:

variables = []

for i, c_lst in enumerate(courses):
    variables.append([])
    v = variables[i]
    for j, c in enumerate(c_lst):
        var = solver.BoolVar(str(c.data["optimizer_id"]))
        v.append(var)
        c.variable = var
```

4.1 Control Panel

We take in some input parameters to define the student's intended course of study.

[14]:
```
# allowed_majors = ["CSCI", "MATH", "ECON", "ENGL"]


# List all intended majors
major_list = ["CSCI", "MATH", "ECON"]

# For math majors: calc_history defines what level of calculus the student␣
 ↪enters williams at.
# 0 indicates student must start at MATH 140, 1 must start at 150/151, 2 means␣
 ↪multi is completed previously.
calc_history = 1

# Study abroad semester should be junior year - semester 5, 6 or both
study_abroad = False
study_abroad_semester = [5]
```

4.2 Constraints

We define constraints for our problem to ensure that the 32-course slate meets all logistical and academic requirements. The first specifies that four courses are taken each semester, unless the student plans to study abroad that semester.

[15]:
```
# Constraint: 4 classes chosen

for sem in range(8):
    if not study_abroad or (sem + 1) not in study_abroad_semester:
        four_classes = solver.Constraint(4, 4)
    else:
        four_classes = solver.Constraint(0, 0)
```

```
    for i, c in enumerate(courses[sem]):
        four_classes.SetCoefficient(c.variable, 1)
```

We implement constraints associated with the early concentration requirements. Freshmen may take at most one course under a given heading at one time; sophomores may take at most two.

[16]:
```
# Early concentration constraints
subjects = ["ARTH", "CSCI", "ECON", "ENGL", "MATH", "STAT"]

early_conc = [1,1,2,2,5,5,5,5]

# Single Semester:
for sem in range(8):
    subject_constraints = {}
    rule = early_conc[sem]
    for s in subjects:
        subject_constraints[s] = solver.Constraint(0,rule)

    for k, c in enumerate(courses[sem]):
        subj = c.data["SUBJECT"]
        subject_constraints[subj].SetCoefficient(c.variable, 1)
        for s in subjects:
            if s != subj:
                subject_constraints[s].SetCoefficient(c.variable, 0)


# First 2 years:
subject_constraints = {}
for s in subjects:
    subject_constraints[s] = solver.Constraint(0,5)

for sem in range(4):
    for k, c in enumerate(courses[sem]):
        subj = c.data["SUBJECT"]
        subject_constraints[subj].SetCoefficient(c.variable, 1)
        for s in subjects:
            if s != subj:
                subject_constraints[s].SetCoefficient(c.variable, 0)
```

We now encode the distribution requirements. At least three courses must be taken in each division over all semesters (at two in each over the first four semesters). Additionally, two WI courses, one DPE course and one QFR course must be taken.

[17]:
```
# Distribution requirements

# First 2 year divisional:

div1_2yr = solver.Constraint(2, solver.infinity())
div2_2yr = solver.Constraint(2, solver.infinity())
```

```
div3_2yr = solver.Constraint(2, solver.infinity())


for sem in range(4):
    for k, c in enumerate(courses[sem]):
        if "DIV_D1" in c.data["WMS_ATTR_SRCH"]:
            div1_2yr.SetCoefficient(c.variable, 1)
            div2_2yr.SetCoefficient(c.variable, 0)
            div3_2yr.SetCoefficient(c.variable, 0)
        elif "DIV_D2" in c.data["WMS_ATTR_SRCH"]:
            div1_2yr.SetCoefficient(c.variable, 0)
            div2_2yr.SetCoefficient(c.variable, 1)
            div3_2yr.SetCoefficient(c.variable, 0)
        elif "DIV_D3" in c.data["WMS_ATTR_SRCH"]:
            div1_2yr.SetCoefficient(c.variable, 0)
            div2_2yr.SetCoefficient(c.variable, 0)
            div3_2yr.SetCoefficient(c.variable, 1)

# 4 year divisional:
div1_4yr = solver.Constraint(3, solver.infinity())
div2_4yr = solver.Constraint(3, solver.infinity())
div3_4yr = solver.Constraint(3, solver.infinity())

for sem in range(8):
    for k, c in enumerate(courses[sem]):
        if "WMS_ATTR_SRCH" not in c.data:
            print(c.data["CATALOG_NBR"], c.data["SUBJECT"])
        if "DIV_D1" in c.data["WMS_ATTR_SRCH"]:
            div1_4yr.SetCoefficient(c.variable, 1)
            div2_4yr.SetCoefficient(c.variable, 0)
            div3_4yr.SetCoefficient(c.variable, 0)
        elif "DIV_D2" in c.data["WMS_ATTR_SRCH"]:
            div1_4yr.SetCoefficient(c.variable, 0)
            div2_4yr.SetCoefficient(c.variable, 1)
            div3_4yr.SetCoefficient(c.variable, 0)
        elif "DIV_D3" in c.data["WMS_ATTR_SRCH"]:
            div1_4yr.SetCoefficient(c.variable, 0)
            div2_4yr.SetCoefficient(c.variable, 0)
            div3_4yr.SetCoefficient(c.variable, 1)


# Distribution Designations:
WI_constraint = solver.Constraint(1, solver.infinity())
DPE_constraint = solver.Constraint(1, solver.infinity())
QFR_constraint = solver.Constraint(1, solver.infinity())
for i, c_lst in enumerate(courses):
    for j, c in enumerate(c_lst):
```

```
        if c.data["WMS_DISTRIB_NT1"] is not None or "WAC" in c.
↪data["WMS_ATTR_SRCH"]:
            WI_constraint.SetCoefficient(c.variable, 1)
        else:
            WI_constraint.SetCoefficient(c.variable, 0)

        if c.data["WMS_DISTRIB_NT2"] is not None or "DPE" in c.
↪data["WMS_ATTR_SRCH"]:
            DPE_constraint.SetCoefficient(c.variable, 1)
        else:
            DPE_constraint.SetCoefficient(c.variable, 0)

        if c.data["WMS_DISTRIB_NT1"] is not None or "QFR" in c.
↪data["WMS_ATTR_SRCH"]:
            QFR_constraint.SetCoefficient(c.variable, 1)
        else:
            QFR_constraint.SetCoefficient(c.variable, 0)
```

We add constraints for each course, specifying that a given instance of each course may be taken at most once (though courses may be listed more than once in the data due to either multiple sections or cross-listings).

[18]:
```
# Cross-listing constraint
xl_courses = set()
for i, c_lst in enumerate(courses):
    for j, c in enumerate(c_lst):
        course_str = f"{c.data['SUBJECT']} {c.data['CATALOG_NBR']}"
        if course_str in xl_courses:
            continue
        xl_courses.add(course_str)
        xl_constraint = solver.Constraint(0,1)
        xl_constraint.SetCoefficient(c.variable, 1)
        this_course_xl = set()
        for i in range(1,5):
            if f"XL Course {i}" in c.data:
                this_course_xl.add(c.data[f"XL Course {i}"])

        for k, x_course_lst in enumerate(courses):
            for l, x_course in enumerate(x_course_lst):
                # Cross-listed as course
                if f"{x_course.data['SUBJECT']} {x_course.data['CATALOG_NBR']}"␣
↪in this_course_xl:
                    xl_constraint.SetCoefficient(x_course.variable, 1)
                # copy of course
                elif f"{x_course.data['SUBJECT']} {x_course.
↪data['CATALOG_NBR']}" == course_str:
                    xl_constraint.SetCoefficient(x_course.variable, 1)
```

We add constraints specifying that scheduling conflicts are not allowed. We do this by specifying that at most one course may be selected whose meeting time includes a given point in time on a given day.

[19]:
```
times = itertools.product(["M", "T", "W", "R", "F"], range(800, 2000, 50))

for sem in range(8):
    for t in times:
        t_constraint = solver.Constraint(0,1)
        for k, c in enumerate(courses[sem]):
            days = c.data["WMS_STND_MTG_PAT1"]
            start_time_str = c.data["WMS_START_TIME1"].replace(":","").
 ↪lstrip("0")
            end_time_str = c.data["WMS_END_TIME1"].replace(":","").lstrip("0")

            if t[0] not in days or start_time_str == "TBA" or start_time_str ==␣
 ↪" ":
                # Doesn't meet that day
                t_constraint.SetCoefficient(c.variable, 0)
                continue
            start_time = int(start_time_str)
            end_time = int(end_time_str)
            if start_time <= t[1] and end_time >= t[1]:
                t_constraint.SetCoefficient(c.variable, 1)
            else:
                t_constraint.SetCoefficient(c.variable, 0)
```

4.3 Prerequisites

Constraints for prereq classes. Currently only enforce prereqs in subjects for which we have majors (CSCI, ECON, ENGL, MATH).

[20]:
```
def single_prereq(target, prereqs):
    """
    Specify that target has the prerequiste prereq
    """
    for sem in range(8):
        sem_target_constraint = solver.Constraint(0, solver.infinity())
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == target[0] and int(c.data["CATALOG_NBR"]) ==␣
 ↪target[1]:
                sem_target_constraint.SetCoefficient(c.variable, -1)
            else:
                sem_target_constraint.SetCoefficient(c.variable, 0)

        for sem_prereq in range(sem):
            for i, c in enumerate(courses[sem_prereq]):
```

```
                if c.data["SUBJECT"] == prereqs[0] and int(c.
 ↪data["CATALOG_NBR"]) == prereqs[1]:
                    sem_target_constraint.SetCoefficient(c.variable, 1)


def inclusive_prereq(target, prereqs):
    """
    Specify that the prerequisite for target is at least one course from the
    list prereqs
    """
    num = len(prereqs)

    for sem in range(8):
        sem_target_constraint = solver.Constraint(0, solver.infinity())
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == target[0] and int(c.data["CATALOG_NBR"]) ==␣
 ↪target[1]:
                sem_target_constraint.SetCoefficient(c.variable, -1)
            else:
                sem_target_constraint.SetCoefficient(c.variable, 0)

        for sem_prereq in range(sem):
            for i, c in enumerate(courses[sem_prereq]):
                for p in prereqs:
                    if c.data["SUBJECT"] == p[0] and int(c.data["CATALOG_NBR"])␣
 ↪== p[1]:
                        sem_target_constraint.SetCoefficient(c.variable, 1)

#csci 19-20
single_prereq(("CSCI", 136), ("CSCI", 134))
single_prereq(("CSCI", 237), ("CSCI", 136))
single_prereq(("CSCI", 256), ("CSCI", 136))
single_prereq(("CSCI", 256), ("MATH", 200))
single_prereq(("CSCI", 361), ("CSCI", 256))
single_prereq(("CSCI", 334), ("CSCI", 237))
single_prereq(("CSCI", 331), ("CSCI", 237))
single_prereq(("CSCI", 338), ("CSCI", 136))
single_prereq(("CSCI", 338), ("CSCI", 237))
single_prereq(("CSCI", 343), ("CSCI", 136))
single_prereq(("CSCI", 356), ("CSCI", 256))
single_prereq(("CSCI", 376), ("CSCI", 136))
single_prereq(("CSCI", 326), ("CSCI", 136))
single_prereq(("CSCI", 333), ("CSCI", 237))
single_prereq(("CSCI", 333), ("CSCI", 136))
single_prereq(("CSCI", 339), ("CSCI", 136))
single_prereq(("CSCI", 339), ("CSCI", 237))
single_prereq(("CSCI", 374), ("CSCI", 256))
```

```
single_prereq(("CSCI", 374), ("CSCI", 136))
single_prereq(("CSCI", 374), ("CSCI", 136))
inclusive_prereq(("CSCI", 315), [("MATH", 150), ("MATH", 151), ("PHYS", 210),␣
 ↪("MATH", 210)])
inclusive_prereq(("CSCI", 315), [("CHEM", 151), ("CHEM", 153), ("CHEM", 155)])


#csci 18-19
single_prereq(("CSCI", 432), ("CSCI", 237))
inclusive_prereq(("CSCI", 432), [("CSCI", 334), ("CSCI", 256)])
single_prereq(("CSCI", 434), ("CSCI", 237))
single_prereq(("CSCI", 434), ("CSCI", 256))


#math 19-20
inclusive_prereq(("MATH", 250), [("MATH", 150) ,("MATH", 151), ("MATH", 200)])
single_prereq(("MATH", 317), ("MATH", 250))
inclusive_prereq(("MATH", 250), [("MATH", 150) ,("MATH", 151), ("MATH", 200)])
single_prereq(("MATH", 325), ("MATH", 250))
single_prereq(("MATH", 341), ("MATH", 250))
single_prereq(("MATH", 350), ("MATH", 250))
inclusive_prereq(("MATH", 350), [("MATH", 150) ,("MATH", 151)])
single_prereq(("MATH", 350), ("MATH", 250))
inclusive_prereq(("MATH", 351), [("MATH", 150) ,("MATH", 151)])
single_prereq(("MATH", 351), ("MATH", 250))
single_prereq(("MATH", 355), ("MATH", 250))
single_prereq(("MATH", 361), ("CSCI", 256))
inclusive_prereq(("MATH", 404), [("MATH", 350) ,("MATH", 351)])
inclusive_prereq(("MATH", 404), [("MATH", 341) ,("STAT", 201)])
single_prereq(("MATH", 422), ("MATH", 355))
single_prereq(("MATH", 426), ("MATH", 350))
single_prereq(("MATH", 482), ("MATH", 355))
inclusive_prereq(("MATH", 485), [("MATH", 350) ,("MATH", 351)])

inclusive_prereq(("MATH", 309), [("MATH", 150) ,("MATH", 151)])
single_prereq(("MATH", 309), ("MATH", 250))
single_prereq(("MATH", 314), ("MATH", 250))
single_prereq(("MATH", 328), ("MATH", 250))
single_prereq(("MATH", 328), ("MATH", 200))
single_prereq(("MATH", 427), ("MATH", 250))
single_prereq(("MATH", 427), ("MATH", 355))
single_prereq(("MATH", 428), ("MATH", 250))
single_prereq(("MATH", 428), ("MATH", 355))
single_prereq(("MATH", 484), ("MATH", 355))
single_prereq(("MATH", 419), ("MATH", 355))

#MATH 18-19
```

```
single_prereq(("MATH", 307), ("MATH", 250))
single_prereq(("MATH", 313), ("MATH", 250))
single_prereq(("MATH", 331), ("MATH", 250))
inclusive_prereq(("MATH", 374), [("MATH", 350) ,("MATH", 351)])
inclusive_prereq(("MATH", 407), [("MATH", 350) ,("MATH", 351)])
single_prereq(("MATH", 407), ("MATH", 355))
inclusive_prereq(("MATH", 433), [("MATH", 250) ,("MATH", 309)])
inclusive_prereq(("MATH", 150), [("MATH", 151) ,("MATH", 351)])
inclusive_prereq(("MATH", 310), [("MATH", 209) ,("MATH", 309)])
single_prereq(("MATH", 310), ("MATH", 250))
single_prereq(("MATH", 321), ("MATH", 250))
inclusive_prereq(("MATH", 334), [("MATH", 200) ,("MATH", 250)])
inclusive_prereq(("MATH", 403), [("MATH", 350) ,("MATH", 351)])
inclusive_prereq(("MATH", 402), [("MATH", 350) ,("MATH", 351)])
single_prereq(("MATH", 411), ("MATH", 355))
single_prereq(("MATH", 458), ("MATH", 200))
single_prereq(("MATH", 458), ("MATH", 355))
inclusive_prereq(("MATH", 459), [("MATH", 209) ,("MATH", 210), ("PHYS", 210),␣
 ↪("MATH", 309)])
single_prereq(("MATH", 487), ("MATH", 355))


#econ
inclusive_prereq(("ECON", 255), [("STAT", 161), ("STAT", 201), ("STAT", 202)])
single_prereq(("ECON", 120), ("ECON", 110))
single_prereq(("ECON", 205), ("ECON", 110))
single_prereq(("ECON", 212), ("ECON", 110))
single_prereq(("ECON", 238), ("ECON", 110))
single_prereq(("ECON", 238), ("ECON", 120))
single_prereq(("ECON", 251), ("ECON", 110))
single_prereq(("ECON", 251), ("MATH", 130))
single_prereq(("ECON", 252), ("ECON", 110))
single_prereq(("ECON", 252), ("ECON", 120))
single_prereq(("ECON", 252), ("MATH", 130))
single_prereq(("ECON", 255), ("MATH", 130))
inclusive_prereq(("ECON", 255), [("STAT", 161), ("STAT", 201), ("STAT", 202)])
single_prereq(("ECON", 299), ("ECON", 110))
single_prereq(("ECON", 299), ("ECON", 120))
inclusive_prereq(("ECON", 299), [("PSCI", 201), ("PSCI", 202), ("PSCI", 203),␣
 ↪("PSCI", 204)])
single_prereq(("ECON", 348), ("ECON", 255))
single_prereq(("ECON", 348), ("POEC", 253))
inclusive_prereq(("ECON", 364), [("ECON", 251), ("ECON", 252)])
inclusive_prereq(("ECON", 364), [("ECON", 255), ("STAT", 201)])
single_prereq(("ECON", 371), ("ECON", 252))
inclusive_prereq(("ECON", 371), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 378), ("ECON", 251))
```

```
single_prereq(("ECON", 378), ("ECON", 252))
inclusive_prereq(("ECON", 378), [("ECON", 255), ("STAT", 346)])
inclusive_prereq(("ECON", 394), [("ECON", 251), ("ECON", 252)])
inclusive_prereq(("ECON", 394), [("ECON", 255), ("POEC", 253), ("STAT", 346)])
single_prereq(("ECON", 451), ("ECON", 251))
single_prereq(("ECON", 451), ("ECON", 252))
inclusive_prereq(("ECON", 451), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 463), ("ECON", 251))
single_prereq(("ECON", 463), ("ECON", 252))
inclusive_prereq(("ECON", 463), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 470), ("ECON", 251))
single_prereq(("ECON", 470), ("ECON", 255))
single_prereq(("ECON", 472), ("ECON", 251))
single_prereq(("ECON", 472), ("ECON", 252))

single_prereq(("ECON", 213), ("ECON", 110))
single_prereq(("ECON", 215), ("ECON", 110))
single_prereq(("ECON", 229), ("ECON", 110))
single_prereq(("ECON", 257), ("ECON", 110))
inclusive_prereq(("ECON", 352), [("ECON", 255), ("POEC", 253)])
single_prereq(("ECON", 357), ("ECON", 251))
inclusive_prereq(("ECON", 357), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 360), ("ECON", 252))
single_prereq(("ECON", 360), ("ECON", 255))
single_prereq(("ECON", 362), ("ECON", 251))
single_prereq(("ECON", 366), ("ECON", 251))
single_prereq(("ECON", 366), ("ECON", 255))
single_prereq(("ECON", 390), ("ECON", 252))
single_prereq(("ECON", 390), ("ECON", 255))
single_prereq(("ECON", 453), ("ECON", 251))
inclusive_prereq(("ECON", 453), [("ECON", 255), ("POEC", 253)])
single_prereq(("ECON", 456), ("ECON", 251))
single_prereq(("ECON", 456), ("ECON", 255))
single_prereq(("ECON", 468), ("ECON", 251))
single_prereq(("ECON", 468), ("ECON", 255))
single_prereq(("ECON", 471), ("ECON", 371))
single_prereq(("ECON", 477), ("ECON", 251))
inclusive_prereq(("ECON", 477), [("ECON", 255), ("STAT", 346)])

#econ 19-20
single_prereq(("ECON", 227), ("ECON", 110))
single_prereq(("ECON", 233), ("ECON", 110))
single_prereq(("ECON", 238), ("ECON", 110))
single_prereq(("ECON", 238), ("ECON", 120))
single_prereq(("ECON", 377), ("ECON", 251))
inclusive_prereq(("ECON", 377), [("ECON", 255), ("POEC", 253)])
single_prereq(("ECON", 451), ("ECON", 251))
```

```
single_prereq(("ECON", 451), ("ECON", 252))
inclusive_prereq(("ECON", 451), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 476), ("ECON", 251))
inclusive_prereq(("ECON", 476), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 214), ("ECON", 110))
single_prereq(("ECON", 345), ("ECON", 251))
single_prereq(("ECON", 345), ("ECON", 252))
inclusive_prereq(("ECON", 345), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 362), ("ECON", 251))
single_prereq(("ECON", 385), ("ECON", 251))
single_prereq(("ECON", 385), ("MATH", 150))
single_prereq(("ECON", 459), ("ECON", 251))
single_prereq(("ECON", 459), ("ECON", 252))
inclusive_prereq(("ECON", 459), [("ECON", 255), ("STAT", 346)])
single_prereq(("ECON", 470), ("ECON", 251))
single_prereq(("ECON", 470), ("ECON", 255))


# 500 level
single_prereq(("ECON", 545), ("ECON", 251))
single_prereq(("ECON", 545), ("ECON", 252))
inclusive_prereq(("ECON", 545), [("ECON", 255), ("STAT", 346)])
inclusive_prereq(("ECON", 510), [("ECON", 255), ("POEC", 253)])
single_prereq(("ECON", 513), ("ECON", 252))
single_prereq(("ECON", 513), ("ECON", 360))
inclusive_prereq(("ECON", 513), [("ECON", 255), ("STAT", 346)])
inclusive_prereq(("ECON", 515), [("ECON", 505), ("ECON", 506)])
single_prereq(("ECON", 516), ("ECON", 251))
single_prereq(("ECON", 516), ("ECON", 255))

inclusive_prereq(("ECON", 523), [("ECON", 110), ("ECON", 504)])
inclusive_prereq(("ECON", 523), [("ECON", 502), ("ECON", 503), ("ECON", 255)])
single_prereq(("ECON", 519), ("ECON", 251))
single_prereq(("ECON", 519), ("ECON", 255))

inclusive_prereq(("ECON", 514), [("ECON", 110), ("ECON", 504)])
inclusive_prereq(("ECON", 514), [("ECON", 502), ("ECON", 503), ("ECON", 255)])
inclusive_prereq(("ECON", 389), [("ECON", 110), ("ECON", 504)])
inclusive_prereq(("ECON", 389), [("ECON", 502), ("ECON", 503), ("ECON", 255)])
single_prereq(("ECON", 536), ("ECON", 252))
single_prereq(("ECON", 536), ("ECON", 255))
single_prereq(("ECON", 501), ("ECON", 251))
single_prereq(("ECON", 501), ("ECON", 252))
inclusive_prereq(("ECON", 501), [("ECON", 255), ("STAT", 346)])

single_prereq(("ECON", 504), ("ECON", 110))
```

```
inclusive_prereq(("ECON", 504), [("ECON", 255), ("STAT", 346), ("ECON", 502),␣
 ↪("ECON", 503)])
single_prereq(("ECON", 505), ("ECON", 251))
single_prereq(("ECON", 505), ("ECON", 252))

#courses with prereq as any ECON: 204, 240,255
#consent of instructor ECON courses: 397, 491, 493, 398, 492, 494


inclusive_prereq(("ENGL", 387), [("ENGL", 203), ("ENGL", 204)])
single_prereq(("ENGL", 382), ("ENGL", 281))
inclusive_prereq(("ENGL", 385), [("ENGL", 283), ("ENGL", 384)])

engl_100_required = [227, 230, 240, 262, 269, 272,
309, 310, 312, 313, 318, 323, 335, 347, 350, 355, 361, 366, 370,
372, 389, 205, 206, 213, 218, 226, 229, 246, 250, 254, 257, 258,
263, 303, 315, 321, 322, 336, 343, 360, 363, 239, 244, 245, 289,
305, 308, 325, 331, 367, 378, 390, 395, 204, 222, 228, 233, 238,
246, 264, 275, 307, 320, 338, 339, 340, 353, 373, 374, 380]

engl_200_required = [371]
engl_300_required = [415, 407, 421, 483]


engl_100_constraint = solver.Constraint(0, solver.infinity())
engl_200_constraint = solver.Constraint(0, solver.infinity())
engl_300_constraint = solver.Constraint(0, solver.infinity())


for sem in range(8):
    for i, c in enumerate(courses[sem]):
        if c.data["SUBJECT"] == "ENGL":
            num = int(c.data["CATALOG_NBR"])
            if num in engl_100_required:
                engl_100_constraint.SetCoefficient(c.variable, -1)
            if num in engl_200_required:
                engl_200_constraint.SetCoefficient(c.variable, -1)
            if num in engl_300_required:
                engl_300_constraint.SetCoefficient(c.variable, -1)
            if num >= 100 and num < 200:
                engl_100_constraint.SetCoefficient(c.variable, 1000)
            if num >= 200 and num < 300:
                engl_200_constraint.SetCoefficient(c.variable, 1000)
            if num >= 300 and num < 393:
                engl_300_constraint.SetCoefficient(c.variable, 1000)
```

4.4 Preclusions

Some courses preclude one from taking some class in the future. For example, you cannot take MATH 150 and then MATH 130.

[21]:
```
def course_precludes(target, exclusion):
    for sem in range(8):
        sem_target_constraint = solver.Constraint(0, 1)
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == exclusion[0] and int(c.data["CATALOG_NBR"])␣
↪== exclusion[1]:
                sem_target_constraint.SetCoefficient(c.variable, 1)
            else:
                sem_target_constraint.SetCoefficient(c.variable, 0)

        for sem_prereq in range(sem):
            for i, c in enumerate(courses[sem_prereq]):
                if c.data["SUBJECT"] == target[0] and int(c.
↪data["CATALOG_NBR"]) == target[1]:
                    sem_target_constraint.SetCoefficient(c.variable, 1)

course_precludes(("MATH", 150), ("MATH", 130))
course_precludes(("MATH", 151), ("MATH", 130))
course_precludes(("MATH", 250), ("MATH", 130))
course_precludes(("MATH", 140), ("MATH", 130))
course_precludes(("MATH", 150), ("MATH", 140))
course_precludes(("MATH", 151), ("MATH", 140))
course_precludes(("MATH", 250), ("MATH", 140))
course_precludes(("MATH", 150), ("MATH", 151))
course_precludes(("MATH", 150), ("MATH", 151))
course_precludes(("STAT", 201), ("STAT", 101))
course_precludes(("STAT", 201), ("STAT", 161))
```

4.5 Major

Each of the following cells defines a function representing a particular major. That function defines constraints for the solver that enforce all of the requirements of that major.

[22]:
```
def cs_major():
    required_course_constraint = solver.Constraint(6, 6)
    required_cs_courses = [134, 136, 237, 256, 334, 361]

    cs_course_constraint = solver.Constraint(8, solver.infinity())

    math_200_constraint = solver.Constraint(1,1)

    math_course_constraint = solver.Constraint(1, solver.infinity())
```

```
    for sem in range(8):
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == "CSCI":
                if int(c.data["CATALOG_NBR"]) in required_cs_courses:
                    required_course_constraint.SetCoefficient(c.variable, 1)
                else:
                    required_course_constraint.SetCoefficient(c.variable, 0)
                if int(c.data["CATALOG_NBR"]) >= 134 and int(c.
↪data["CATALOG_NBR"]) < 493:
                    cs_course_constraint.SetCoefficient(c.variable, 1)
                else:
                    cs_course_constraint.SetCoefficient(c.variable, 0)

            if c.data["SUBJECT"] == "MATH":
                if int(c.data["CATALOG_NBR"]) == 200:
                    math_200_constraint.SetCoefficient(c.variable, 1)
                elif int(c.data["CATALOG_NBR"]) > 200 and int(c.
↪data["CATALOG_NBR"]) < 490:
                    math_course_constraint.SetCoefficient(c.variable, 1)
```

[23]:
```
def create_math_major(completed):
    def math_major():
        core_course_constraint = solver.Constraint(3, 3)
        core_courses = [250, 350, 355]

        applied_course_constraint = solver.Constraint(1, solver.infinity())
        applied_math_courses = [200, 209, 210]
        applied_stat_courses = [201, 231]

        math_course_constraint = solver.Constraint(9, solver.infinity())

        capstone_constraint = solver.Constraint(1, solver.infinity())

        if completed <= 0:
            calc_constraint = solver.Constraint(1, 1)
            min = 140
            single_prereq(("MATH", 150), ("MATH", 140))
            single_prereq(("MATH", 151), ("MATH", 140))
        if completed <= 1:
            multi_constraint = solver.Constraint(1,1)
            min = 150
        if completed > 1:
            min = 152


        for sem in range(8):
            for i, c in enumerate(courses[sem]):
```

```
                if c.data["SUBJECT"] == "MATH":
                    if int(c.data["CATALOG_NBR"]) in core_courses:
                        core_course_constraint.SetCoefficient(c.variable, 1)
                    if int(c.data["CATALOG_NBR"]) in applied_math_courses:
                        applied_course_constraint.SetCoefficient(c.variable, 1)
                    if int(c.data["CATALOG_NBR"]) >= min and int(c.
 ↪data["CATALOG_NBR"]) < 493:
                        math_course_constraint.SetCoefficient(c.variable, 1)

                    if int(c.data["CATALOG_NBR"]) >= 400 and int(c.
 ↪data["CATALOG_NBR"]) < 493:
                        capstone_constraint.SetCoefficient(c.variable, 1)

                    if completed <= 0:
                        if int(c.data["CATALOG_NBR"]) == 140:
                            calc_constraint.SetCoefficient(c.variable, 1)
                    if completed <= 1:
                        if int(c.data["CATALOG_NBR"]) in [150, 151]:
                            multi_constraint.SetCoefficient(c.variable, 1)

                elif c.data["SUBJECT"] == "STAT":
                    if int(c.data["CATALOG_NBR"]) in applied_stat_courses:
                        applied_course_constraint.SetCoefficient(c.variable, 1)

    return math_major

# Pass 0, 1 or 2 into function
# 0 indicates student must start at MATH 140, 1 must start at 150/151, 2 means␣
 ↪multi is completed previously
```

[24]:
```
def econ_major():
    required_course_constraint = solver.Constraint(5, 5)
    required_econ_courses = [110,120,251,252,255]

    elective_constraint = solver.Constraint(4, solver.infinity())

    elective_300 = solver.Constraint(2, solver.infinity())
    elective_400 = solver.Constraint(1, solver.infinity())


    for sem in range(8):
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == "ECON":
                if int(c.data["CATALOG_NBR"]) in required_econ_courses:
                    required_course_constraint.SetCoefficient(c.variable, 1)
                if int(c.data["CATALOG_NBR"]) >= 300 and int(c.
 ↪data["CATALOG_NBR"]) <= 395:
```

```
                    elective_300.SetCoefficient(c.variable, 1)
                    elective_constraint.SetCoefficient(c.variable, 1)
                elif int(c.data["CATALOG_NBR"]) >= 450 and int(c.
↪data["CATALOG_NBR"]) <= 480:
                    elective_400.SetCoefficient(c.variable, 1)
                    elective_constraint.SetCoefficient(c.variable, 1)
                elif int(c.data["CATALOG_NBR"]) >= 200 and int(c.
↪data["CATALOG_NBR"]) <= 299:
                    elective_constraint.SetCoefficient(c.variable, 1)
```

[25]:
```
def engl_major():
    course_100 = solver.Constraint(1, solver.infinity())
    gateway_200 = solver.Constraint(1, solver.infinity())
    criticism = solver.Constraint(1, solver.infinity())
    elective_300 = solver.Constraint(3, solver.infinity())
    history = solver.Constraint(3, solver.infinity())

    gateway_courses_engl = [205,206,230,245,254,289, 209, 218, 222, 228, 233,␣
↪238, 246, 258, 275,
                            240, 249, 262, 269, 218, 226, 229, 258, 263]

    criticism_courses_engl = [113, 230, 240, 262, 309, 318, 322, 323, 335, 370,␣
↪445, 456, 246,
                              321, 322, 323, 339, 340, 363, 402, 415, 266, 335,␣
↪367, 376, 395,
                              407, 421, 209, 238, 246, 321, 322, 339, 340, 483]
    history_courses_engl = [269, 310, 372, 445, 227, 240, 313, 318, 322, 323,␣
↪338, 347, 115, 262,
                            269, 272, 312, 350, 318, 366, 387, 389, 201, 303,␣
↪315, 321, 339,226,
                            246, 258, 263, 322, 323, 402, 202, 206, 216, 218,␣
↪220, 226, 229, 312,
                            336, 340, 360, 402]



    for sem in range(8):
        for i, c in enumerate(courses[sem]):
            if c.data["SUBJECT"] == "ENGL":
                num = int(c.data["CATALOG_NBR"])
                if num >= 100 and num <= 199:
                    course_100.SetCoefficient(c.variable, 1)
                if num in gateway_courses_engl:
                    gateway_200.SetCoefficient(c.variable, 1)
                if num in criticism_courses_engl:
                    criticism.SetCoefficient(c.variable, 1)
                if (num >= 300 and num < 393) or (num >= 400 and num < 493):
```

```
                    elective_300.SetCoefficient(c.variable, 1)
                if num in history_courses_engl:
                    history.SetCoefficient(c.variable, 1)
```

Having defined the constraints for each major, we now apply them for each major that is designated in the control panel.

[26]:
```
major_functions = {
    "CSCI": cs_major,
    "MATH": create_math_major(calc_history),
    "ECON": econ_major,
    "ENGL": engl_major
}
for m in major_list:
    major_functions[m]()
```

5 Objective Function

Here we define our objective function. Our objective function seeks to maximize the percentage of factrak reviewers who say they would recommend the course to a friend. However, for data integrity purposes, this version of the notebook does not come with factrak data. Therefore, this objective function will minimize course catalog number.

[27]:
```
# # Weight options:

# # Evaluation type
# paper = 0
# exam = 0
# attendance = 0

# # Component
# lecture = 0
# seminar = 0
# tutorial = 0

# # TODO: incorporate these into objective
```

[28]:
```
# objective = solver.Objective()

# This section is commented out because this version of the notebook will not␣
 ↪be accompanied by factrak data.

# for i, c_lst in enumerate(courses):
#     for j, c in enumerate(c_lst):
#         course_str = f'{c.data["SUBJECT"]} {c.data["CATALOG_NBR"]}'
#         if course_str not in factrak_survey_data["recommend"]:
#             objective.SetCoefficient(c.variable, .75)
```

```
#             #overall average rec rate is .85, giving penalty for no rec data
#         else:
#             rec = int(factrak_survey_data["recommend"][course_str])
#             objective.SetCoefficient(c.variable, rec)

# # Specify that we want to maximize this objective function
# objective.SetMaximization()
```

[29]:
```
# Alternate objective function seeks to minimize course catalog numbers.
objective = solver.Objective()
for i, c_lst in enumerate(courses):
    for j, c in enumerate(c_lst):
        objective.SetCoefficient(c.variable, int(c.data["CATALOG_NBR"]))

# Specify that we want to minimize this objective function
objective.SetMinimization()
```

[30]:
```
# Very satisfying line of code:
solver.Solve()
```

[30]:
```
0
```

5.1 Solution

Solving the problem gives our solution of 32 courses:

[32]:
```
for i,v_lst in enumerate(variables):
    print(f"\nSemester {i+1}:")
    for j, v in enumerate(v_lst):
        if v.solution_value():
            print(f"{courses[i][j].data['SUBJECT']} {courses[i][j].
 ↪data['CATALOG_NBR']}")
```

```
Semester 1:
CSCI 134
ECON 110
MATH 200
STAT 101

Semester 2:
ARTH 102
CSCI 136
ECON 120
MATH 130

Semester 3:
CSCI 237
MATH 151
```

```
MATH 250
STAT 161

Semester 4:
ARTH 103
CSCI 334
ECON 251
ECON 252

Semester 5:
CSCI 326
ECON 255
MATH 307
MATH 350

Semester 6:
CSCI 256
ECON 352
MATH 210
MATH 403

Semester 7:
ARTH 101
CSCI 331
ECON 451
MATH 102

Semester 8:
CSCI 361
ECON 345
MATH 309
MATH 355
```

6 Extensions

This project is a proof-of-concept; there are many ways in which we could extend it further, even with the data that we currently have access to. The first and most obvious is the generalize this across the entire course catalog. We also have data, both from the catalog and from factrak, that could be usefully incorporated into the objective function. * Course component - student may choose an objective function that emphasizes seminar, lecutre or tutorial courses * Evaluation type - can weight courses that have 'paper' or 'exam' in the evaluation description * Additional factrak statistics - Factrak includes ratings on a number of different dimensions in addition to the student recommendations