Knowledge Graph Querying for Course Schedule Building

Adebayo Braimah
Stony Brook University
ID Number: 115099306

May 10, 2024

1 Problem & Plan

1.1 Problem Description

Scheduling is a highly non-trivial problem that arises in throughout many fields and industrial applications. Several examples include the job-scheduling problem [2], and the nurse scheduling problem [3]. University course planning/scheduling and understanding of graduation requirements can be a difficult process for new, in-coming students at all levels of one's education [1]. Generally, new students are guided through this process by way of an academic advisor. However, this approach is expensive in both time and personnel (which are usually university faculty) – especially in the case in which the personnel have to be trained on where to find and understand these graduation requirements. Additionally, in some cases – the advising can be further complicated by the student's own personal interests (e.g. research focus, specific areas of interests, etc). The proposed solution to this problem would be knowledge graphs that are queried for course schedule building. A summary of the inputs and outputs are shown below:

1.1.1 Input & Output

Inputs:

- Major
- Degree level: bachelors (masters and doctorate are not implemented in this project)
- Current degree progress (e.g. classes already taken)
- Knowledge graphs
    - Graduation requirements
    - Department policies (e.g. restrictions on pass/fail courses)

Outputs:

- Recommended schedule(s)
- Course recommendations

1.1.2 Requirements

The requirements for this project would include:

- Tools
    - Python
        - Selenium (for web scraping, and web browser interface)
        - BeautifulSoup4 (for web scraping)
        - requests (for web scraping)
        - Pandas [6] (for organizing data)
    - Clingo (ASP[1] solver) [5]
- Performance evaluation
    - Measure the time and accuracy of each query and compare it to Stony Brook University's schedule builder [4]
- Comparison of solutions
    - Compare the output of the course schedules and recommendations with Stony Brook University graduation requirements (for computer science majors).

1.1.3 Example use-cases

Moreover, the use cases of these solutions from this project is widely applicable to Stony Brook University's undergraduate student population as a whole. For example, these groups of students would find significant utility from this project's solutions:

- An undergraduate computer science student looking to meet the graduation requirements for a combined BS/MS in 5 years.
- A BS graduate student in the computer science department looking to satisfy graduation requirements in 4 years, taking 12–15 credits per semester.

Granted the above use-cases were for computer science students – ideally, most of the Stony Brook University student population could derive some benefit from this project.

[1] Answer Set Programming

1.2 State of the art

Current state-of-the-art (SOTA) approaches for this problem at Stony Brook University includes the schedule builder in addition to LUNCH [1]. In the case of LUNCH, while it is a mostly automated approach – it does require significant user input, and only provides the schedule for one semester. In the case of the schedule builder – it will mostly help students build a schedule, semester by semester [4], but it will not recommend/suggest classes.

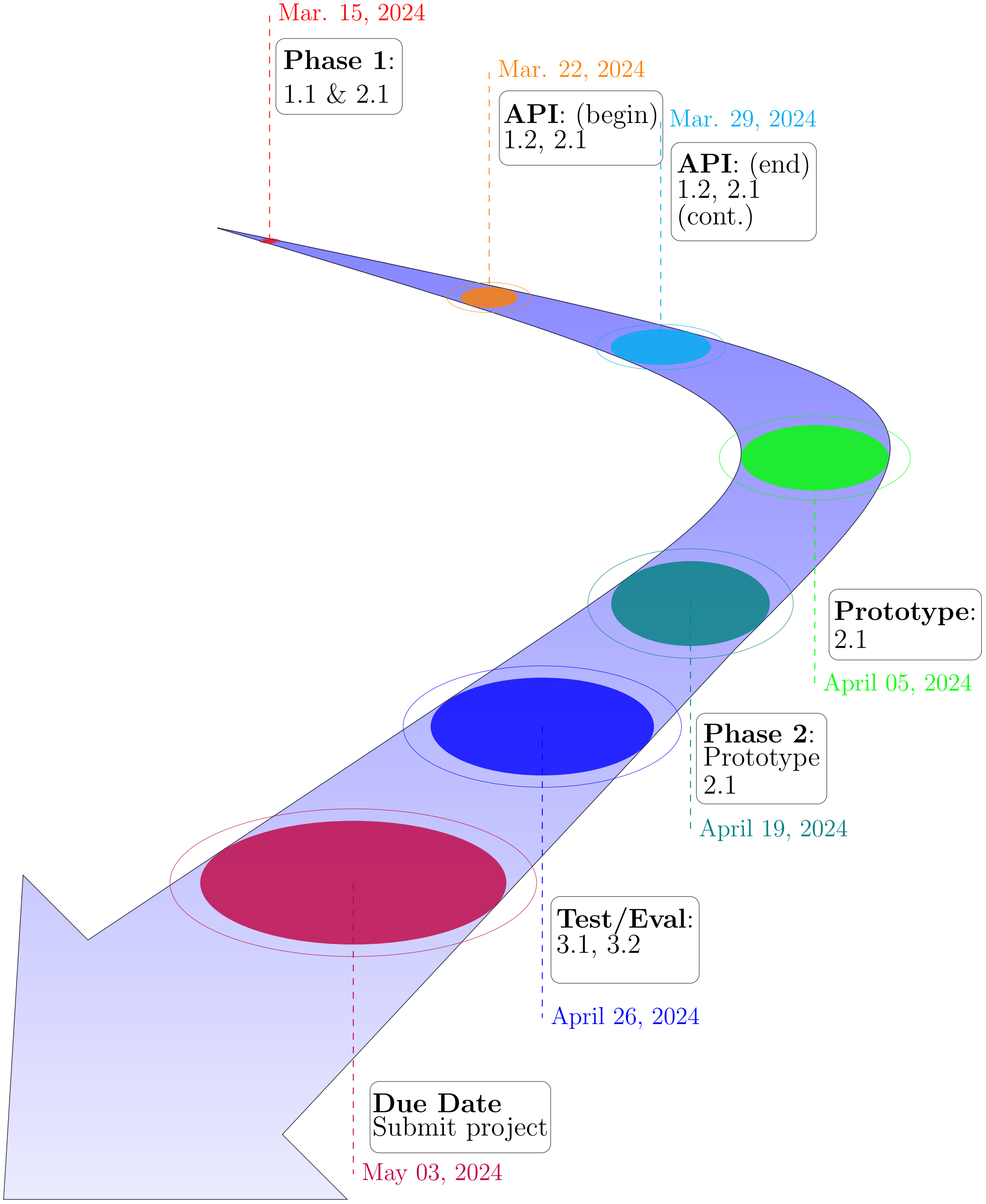
1.3 Tasks & Sub-tasks

Currently, this project has no relation to any external projects (both through adjacent course work and for research purposes). Below are the corresponding tasks and sub-tasks for the project:

- **Task 1**: Knowledge graph construction (via web scraping)
  - **Sub-task 1.1**: Create knowledge graphs of Stony Brook University undergraduate and graduate computer science graduation requirements (including department policies)
- **Task 2**: Build API
  - **Sub-task 2.1**: Create output knowledge base that can be queried.
- **Task 3**: Test & Evaluate
  - **Sub-task 3.1**: Perform and automate test queries using commonly asked questions
  - **Sub-task 3.2**: Evaluate performance (query time and accuracy)

1.4 Project plan

The repository for the planned code base is located at this public GitHub repository. Additionally, the planned timeline of the project is shown below in subsection 1.4 Project plan, with each set of tasks and sub-tasks (subsection 1.3) as checkpoints.



2 Design

2.1 API

The API implementation can be summarized as shown in the UML[2] diagram in Figure 1 in addition to class structures for `KnowledgeBase` and `KnowledgeGraph` with their corresponding class attributes in Figure 2. The overall system design can be described in three major components:

1. Knowledge graph construction from SBU course catalog (described in Figure 3).
2. Converting/preprocessing JSON knowledge graph to Clingo atoms and predicates (described in Figure 4).
3. Query the knowledge base of course requirements(described in Figure 5).

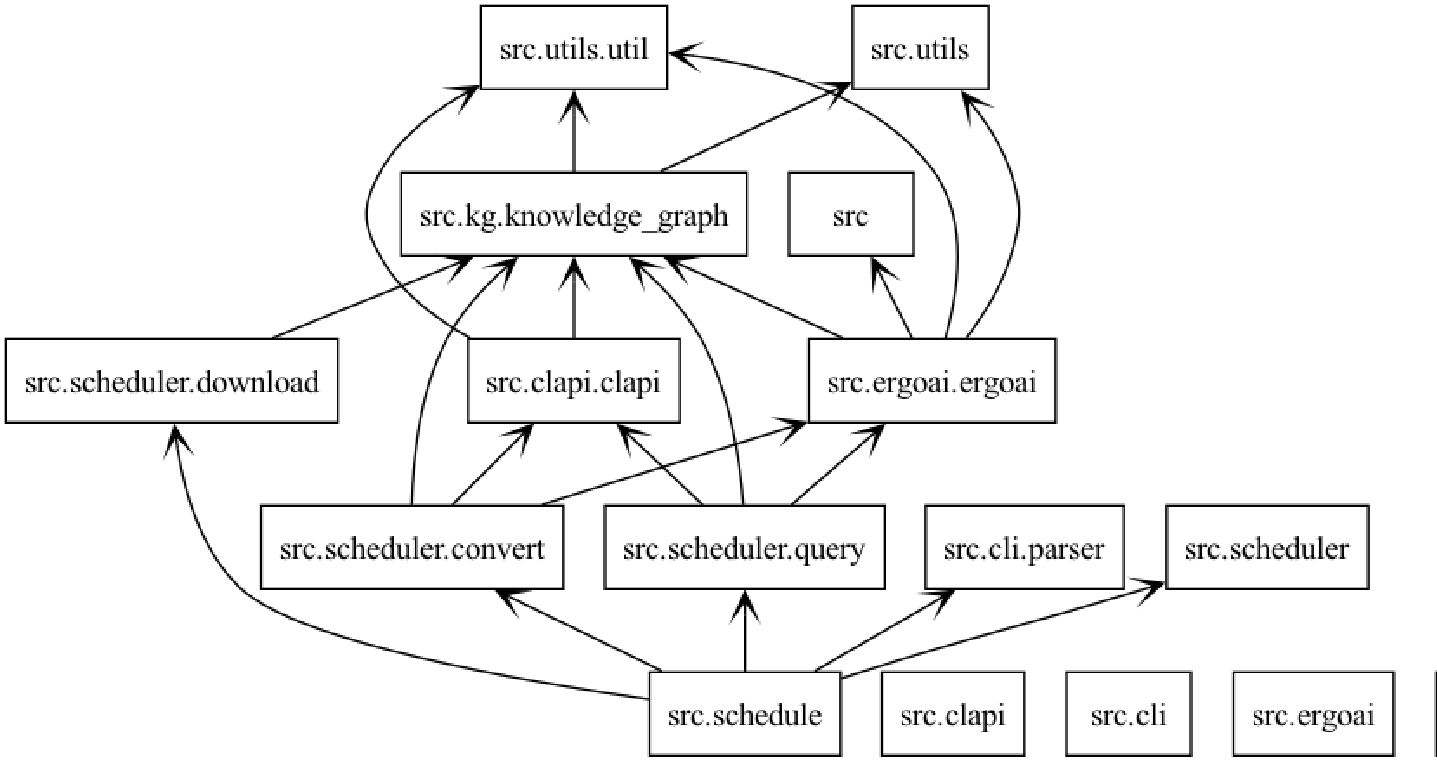


Figure 1: UML Diagram of the python package

The driver program of this project can be run as shown from the command line:

```
./driver.py
```

Note, that file permissions may need to be changed for the file to run.

[2] Unified Modeling Language

The outputs of the file are the results of the test query, printed to the command line:

```
--------------------------------------------
Begin: query_clingo  |  May-11-2024 04:00:08
--------------------------------------------

clingo version 5.7.1
Reading from ...projects/CSE505/results/cse_courses.lp ...
Solving...
Answer: 1
schedule(che129,spring) schedule(che132,spring) schedule(geo122,spring)
schedule(ams110,fall) schedule(ams301,fall) schedule(cse304,fall)
schedule(cse506,fall)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.058s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.053s

--------------------------------------------
End: query_clingo Execution time: 0.07 sec.   |  May-11-2024 04:00:08
--------------------------------------------
```

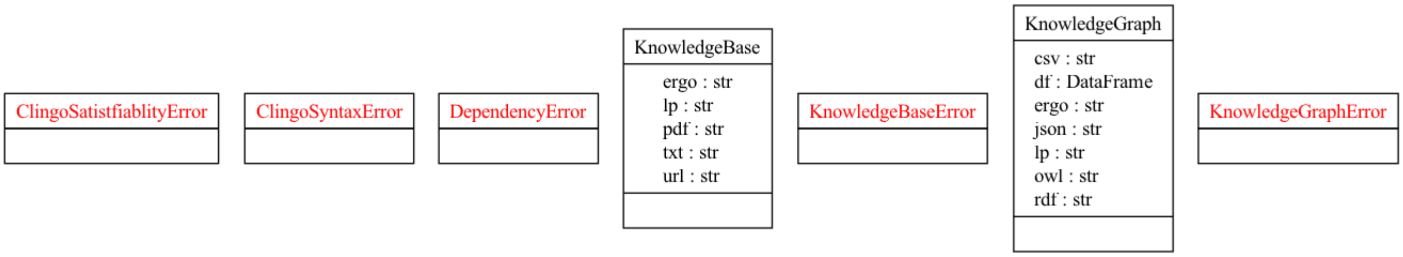


Figure 2: UML Diagram of the python package's classes and corresponding attributes.

Lastly, in the `src/scripts` directory, one can batch download other major's courses information. These set of scripts were mainly included to download and find all of the non-CSE major prerequisite courses (e.g. MAT 123, MATH 125, etc). The scripts `src/scripts/download_courses.py` and `src/scripts/download_courses.sh` accomplish this purpose. The script `src/scripts/download_courses.sh` is a wrapper script that parallelizes `src/scripts/download_courses.py`. However, the wrapper script has an additional external dependency: GNU Parallel [7], which must be installed and on the system path variable for the batch download scripts to work correctly.

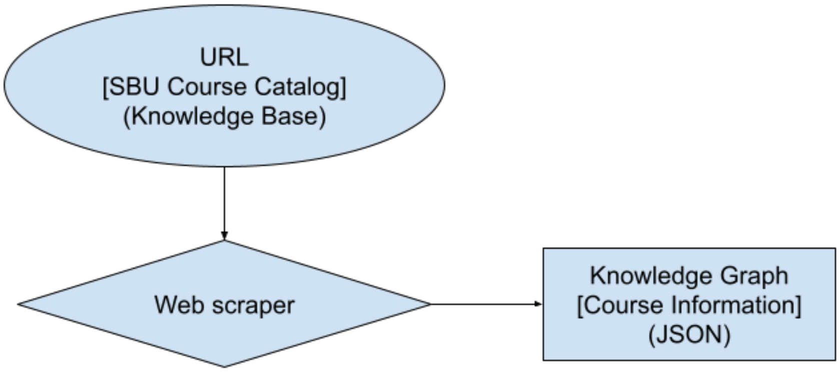


Figure 3: Design flowchart of knowledge base to knowledge graph creation, performed via web scraping of Stony Brook University's online course catalog.

2.1.1 Implementation Details

The implementation details of this project are depicted in the UML diagrams shown in Figures 1 and 2. Moreover, the package `./src/schedule.py`[3] is the main entry point into this program, with a command line interface (CLI) and three sub-commands: `graph`, `convert`, and `query`[4]. These sub-commands are described in more detail below (required arguments only):

- `graph`
  - major ('CSE' for most examples)
- `convert`
  - json-file (The output from the above step)
  - `clingo` (We want the output file to consist of clingo atoms and predicates)
- `query`
  - knowledge (input knowledge base/graph to be queried)
  - query (input query string or file that contains rules/queries to be executed, may be specified multiple times)

2.2 Third Party Libraries

The third party libraries currently used (mentioned in 1.1.1) include Selenium (for web scraping, and browser interface), Pandas (data organization prior to writing knowledge graphs), BeautifulSoup4 (for web scraping) and Clingo (ASP solver, installed via (mini)conda).

[3] NOTE: Should more help be needed, type: `./src/schedule.py -h` for the full help menu.

[4] NOTE: Although `ErgoAI` is available in the option menu, it is not fully supported, especially for query execution.

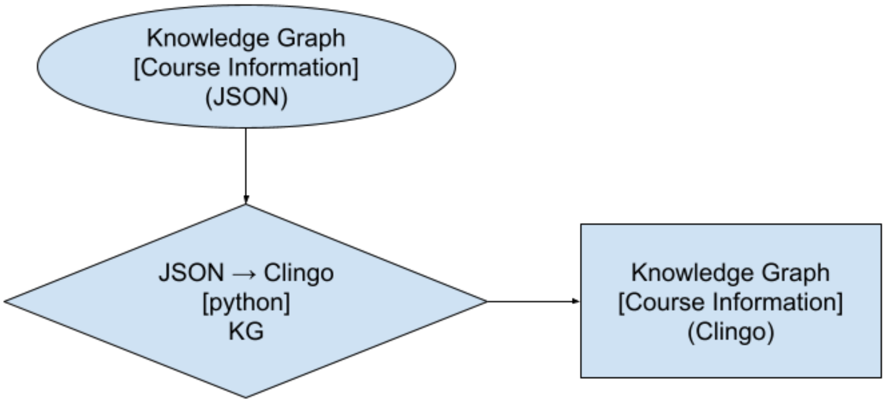


Figure 4: Design flowchart of JSON knowledge graph conversion to a clingo knowledge graph.

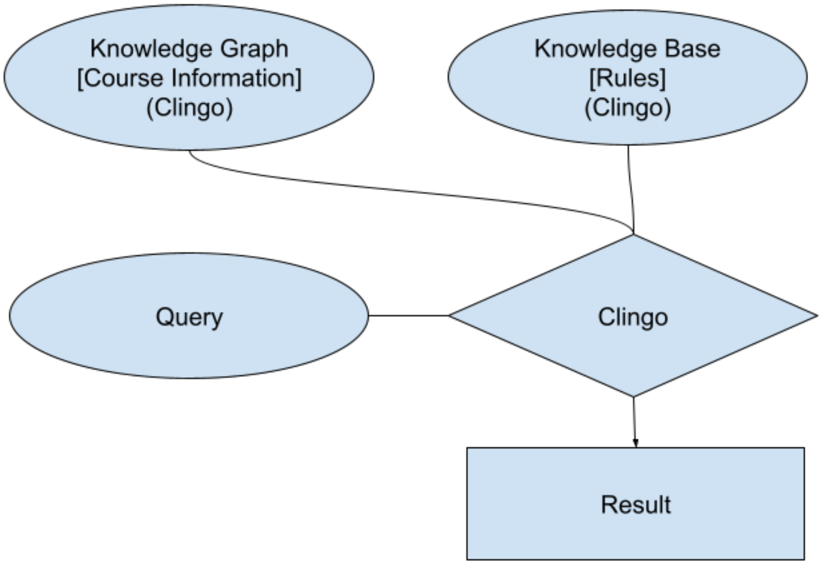


Figure 5: Design flowchart of knowledge base and knowledge graph querying via Clingo.

2.3 Documentation

The documentation of this project is contained in the `doc` folder. Documentation was also written in reStructured Text (`.rst`) files, and built using python via Sphinx (`HTML` documentation can be found in `doc/build/html/index.html`). The `HTML` documentation can found online here (recommended method of viewing this documentation).

2.4 Design Document

The design document[5] for this project is located here in this Google Document. The design document can be briefly summarized as:

- The project purpose: stating the problem, goals and those impacted (see section 1.1.1).
- Project scope: What features will be built, and what features will not be implemented.
- Stakeholders: The target audience, and those impacted by this project.
- Requirements: Functional and non-functional requirements (see section 1.1.2).
- Project timeline and milestones (see section 1.4).
- Architecture and system design (see Figures: 1, 2, 3, 4 & 5)
- Test & quality assurance: Test cases and queries that need to be covered (see section 4).

3 Implementation

3.1 Design Implementation

The current design implementation has focused on translating knowledge graphs into facts. Correspondingly, knowledge bases were translated into rules. For example, the the class information shown in the JSON snippet below, could be translated into the following fact[6]:

```
{
    .
    .
    .
    "CSE214":{
    "CourseTitle":"Data Structures",
    "Career":"Undergraduate",
    "Credits":4.0,
    "Prerequisites":[
        [
            "CSE114"
        ]
    ],
```

[5] Format referenced from this AWS design document in this GitHub repository

[6] NOTE: JSON fields "spring" and "fall" entail if the course is offered in the spring and/or in the fall (i.e. 1 implies offered, 0 implies not offered).

```
    "Antirequisites":"NONE",
    "Corequisites":"NONE",
    "Description":"An extension of programming methodology to data storage and manipulation on complex data sets. Topics include: programming and applications of data structures; stacks, queues, lists, binary trees, heaps, priority queues, balanced trees and graphs. Recursive programming is heavily utilized. Fundamental sorting and searching algorithms are examined along with informal efficiency comparisons.",
    "spring":1.0,
    "fall":1.0
    },
    "CSE215":{
        "CourseTitle":"Foundations of Computer Science",
        "Career":"Undergraduate",
        "Credits":4.0,
        "Prerequisites":[
            [
                "AMS151",
                "MAT125",
                "MAT131"
            ]
        ],
        "Antirequisites":"NONE",
        "Corequisites":"NONE",
        "Description":"Introduction to the logical and mathematical foundations of computer science. Topics include functions, relations, and sets; recursion; elementary logic; and mathematical induction and other proof techniques.",
        "spring":1.0,
        "fall":1.0
    },
.
.
.
}
```

Facts translation in Clingo (located in: `src/resources/cse_courses.lp`)):

```
% Clingo code
% Define course(course_name, credits, career, offered_spring, offered_fall)
course(cse214, 4, "Undergraduate", 1, 1).
course(cse215, 4, "Undergraduate", 1, 1).
.
.
.
% Define prerequisites
.
.
.
% Prerequisites for CSE214
:- course(cse214, 4, "Undergraduate", 1, 1),
   not course(cse114, _, "Undergraduate", _, _).

% Prerequisites for CSE215
:- course(cse215, 4, "Undergraduate", 1, 1),
   not course(ams151, _, "Undergraduate", _, _),
   not course(mat125, _, "Undergraduate", _, _),
   not course(mat131, _, "Undergraduate", _, _).
```

Correspondingly, the knowledge base facts were translated into rules. For example, the fact a student needs take a minimum of 12 credits a semester (to maintain full time student status, shown on line 3 in the listing below as a constant), but a maximum of 18 credits (shown on line 4 as constant in the listing below) over N semesters to graduate with CSE major 80 credits and 120 total credits would be translated as the following in Clingo (in the file, `results/cse_bs_grad_reqs.lp`):

```
 1 % Constants
 2 #const max_semesters = 12.
 3 #const min_credits_per_semester = 12.
 4 #const max_credits_per_semester = 18.
 5
 6 % Define possible semesters
 7 semester(1..max_semesters).
 8
 9 % Schedule courses across semesters
10 1 { schedule(Course, Sem) : semester(Sem) } 1 :- course(Course, Credits,
      Career, Spring, Fall).
11
12 % Prevent scheduling of courses already taken
13 :- course_taken(Course), schedule(Course, _).
14
15 % Semester limits: at least 12 credits and at most 18 credits per semester
16 :- semester(Sem), SemCredits = #count { Credits, Course : schedule(Course,
       Sem), course(Course, Credits, "Undergraduate", _, _) }, SemCredits <
      min_credits_per_semester.
17 :- semester(Sem), SemCredits = #count { Credits, Course : schedule(Course,
       Sem), course(Course, Credits, "Undergraduate", _, _) }, SemCredits >
      max_credits_per_semester.
18
19 % Ensure all scheduled courses comply with seasonal offerings
20 :- schedule(Course, Sem), course(Course, _, "Undergraduate", Spring, Fall)
      ,
21    Sem \ 2 = 1, Fall = 0.
22 :- schedule(Course, Sem), course(Course, _, "Undergraduate", Spring, Fall)
      ,
23    Sem \ 2 = 0, Spring = 0.
24
25 % Count total credits and major credits
26 total_credits(Total) :- Total = #sum { Credits, Course : schedule(Course,
      _), course(Course, Credits, "Undergraduate", _, _) }.
27 major_credits(Major) :- Major = #sum { Credits, Course : schedule(Course,
      _), course(Course, Credits, "Undergraduate", _, _) }.
28
```

```
% Graduation requirements
:- total_credits(Total), Total < 120.
:- major_credits(Major), Major < 80.

% Objective to minimize the number of semesters
#minimize { 1, Sem : schedule(_, Sem) }.

#show schedule/2.
```

Moreover, the above code suffers from combinatorial explosion (i.e. the number of combinations that satisfies the model are so large, that the optimal model would take a significant amount of time to be computed). Instead, computing the schedule of two semesters is far more feasible. The code to do so is located in the file `results/sem.lp`):

```
% Semester definition
semester(spring).
semester(fall).

% Scheduling a course in a semester
{ schedule(Course, spring) : course(Course, Credits, Career, 1,
    OfferedFall) } :-
     not course_taken(Course).

{ schedule(Course, fall) : course(Course, Credits, Career, OfferedSpring,
    1) } :-
     not course_taken(Course).

% Credit calculation per semester
credits_sum(Sem, TotalCredits) :-
     semester(Sem),
     TotalCredits = #sum { Credits, Course : schedule(Course, Sem), course(
         Course, Credits, _, _, _) }.

% Enforce credit limits
:- credits_sum(Sem, TotalCredits), TotalCredits < 12.
:- credits_sum(Sem, TotalCredits), TotalCredits > 18.

% Output directives to facilitate result interpretation
#show schedule/2.
```

3.1.1 Design Issues & Problems

The query written in `results/cse_bs_grad_reqs.lp` suffered from grounding overhead, and combinatorial explosion in Clingo. These two issues in particular were likely caused by the large input of course atoms (about 207 listed undergraduate and graduate courses, in addition to their corresponding pre- and co-requisites). From the combination defined in Eq. 1, and assuming that an undergraduate student could take anywhere between 4 and 6 courses (with the basic assumption that each course averages to about 3 credits), then there would exist anywhere between $1.457 \cdot 10^6$ to $7.430 \cdot 10^7$ possible combinations.

$$\begin{aligned} C(n,k) &= \binom{n}{k} \\ &= \frac{n!}{k!(n-k)!} \end{aligned} \tag{1}$$

With such a significantly large search space of 207 courses – these large number of possible combinations make sense, further clarifying the extremely long runtime of `results/cse_bs_grad_reqs.lp`, for computing all possible course schedules.

3.2 Design Documents (updated)

See section 2.4 for a summary of the design document. The design document can also be found here in this Google Document. The design document at this link has been updated in accordance with the specifications of this project.

3.3 Project Implementation

The project implementation can be found in the following code base, located at this GitHub Repository. The development effort of the project required sizable effort – especially in the parts pertaining to ErgoAI (which is not currently implemented). Below is a brief summary of the project:

- Languages
  - Python
    - Selenium
    - Pandas
    - BeautifulSoup4
    - Requests
  - Clingo
  - ErgoAI (not implemented, but attempted)
- Tools
  - Conda (python environment management)
  - Sphinx (automated documentation)
  - Graphviz & PyLint (for creating UML diagrams)
  - Black (python code formatter)
- Development effort: Fairly sizable, especially in reference to ErgoAI (not implemented).
- Code base size: moderately sized at 8,657 lines of code.

Lastly, this project, in its current state significantly lags behind the SOTA approaches – mainly in terms of reliability (correctness) rather than speed.

4 Testing & Evaluation

The set of tests that have been performed include:

- Unit tests (for utility functions)
- Performance & Correctness (performed via manual creation of course schedule)

The unit tests mainly covered the utility functions and were implemented using PyTest. The unit tests are located in the `src/tests` directory, can be run by typing: `cd src/tests`, and running `pytest` (assuming pytest is already installed). Manual testing was performed to evaluate the full scheduler and the two semester scheduler performance and correctness. The manual method (hand making schedules) was often correct – but more time consuming (in both cases). The automated method for the 2 semester approach is fast – but the correctness is questionable, as the courses are not sensibly scheduled. Additionally, the automated approach for 12 semester scheduling took so long, that the program was terminated prior to finding any models (see sec. 3.1.1 for explanations as why this was the case). Results of the manual testing performance (in seconds) are shown in Figures 6 and 7. Additionally, the commands to perform the (automated) evaluation are as follows:

```
# 2 sem
./src/schedule.py query --knowledge=results/cse_courses.lp --query=results/sem.lp --query=results/cse_prereqs.lp --clingo

# 12 sem approach # NOTE: This will not stop running
./src/schedule.py query --knowledge=results/cse_courses.lp --query=results/cse_bs_grad_reqs.lp --query=results/cse_prereqs.lp --clingo

# eval - 2 sem
./src/schedule.py query --knowledge=results/eval/cse_courses.eval.lp --query=results/sem.lp --query=results/cse_prereqs.lp --clingo

# eval - 12 sem # NOTE: This will not stop running
./src/schedule.py query --knowledge=results/eval/cse_bs_grad_reqs.eval.lp --query=results/eval/cse_courses.eval.lp --query=results/cse_prereqs.lp --clingo
```

5 Challenges & Summary

The main challenges experienced in this project was that SBU's course catalog is not easily accessible for analysis. This was evident by the absence of available API access. Additionally, the course catalog was written in JavaScript – making web scraping a difficult endeavor. Furthermore, satisfying the constraint in clingo for as originally planned was highly improbable (as explained in sec. 3.1.1) – which can mainly be attributed to grounding overhead and combinatorial explosion. The original approach is more appropriate for checking if degree requirements have been satisfied. Lastly, modifying the core idea to accommodate two semesters rather than an arbitrary number of semesters has shown why the scheduling problem in this context is extremely challenging.

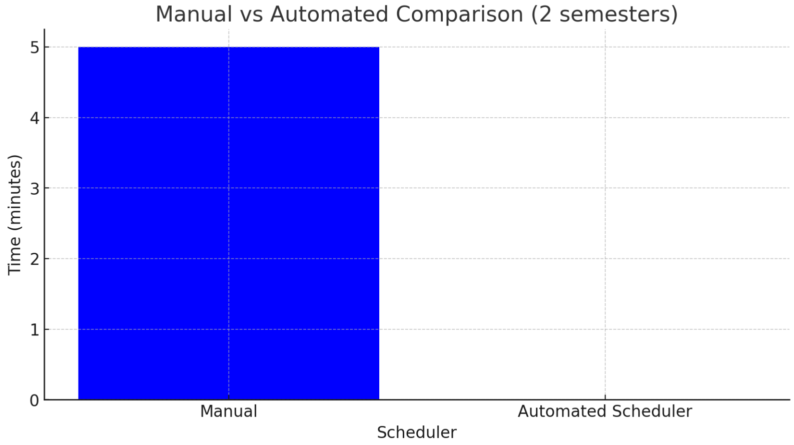


Figure 6: Bar graph depicting the schedulers performance (time in sec.) vs manual testing for the 2 semester scheduling.

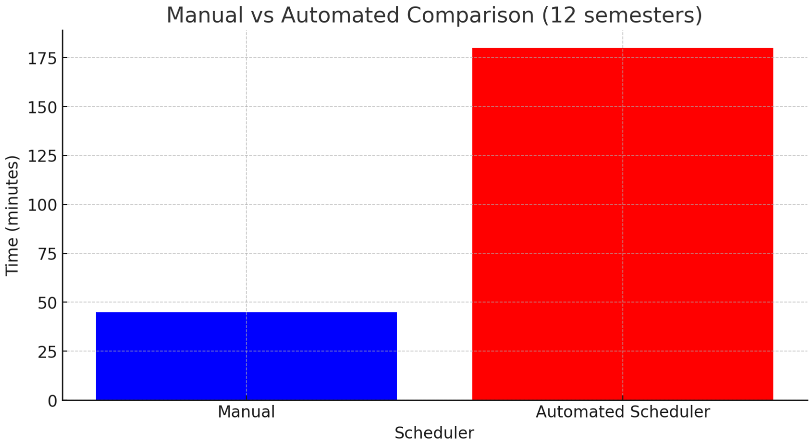


Figure 7: Bar graph depicting the schedulers performance (time in sec.) vs manual testing for the 12 semester scheduling.

6 Acknowledgements

- Prof. Annie Liu for the project idea and insights
- Geoffrey Churchill for discussions and insights
- Prof. Paul Fodor for discussions and insights
- Prof. Michael Kifer for email correspondence and help with ErgoAI
- Gokul and Apeksha for advice on scheduling just one semester to test performance and correctness.

References

[1] Ana Y.F. de Lima, Briza M. D. de Sousa, Daniel P. Cardeal, Jessica Y. N. Sato, Lorenzo B. Salvador, and Bruna Bazaluk. LUNCH: an Answer Set Programming System for Course Scheduling. https://sol.sbc.org.br/index.php/eniac/article/download/25756/25572/, 2023. ACCESSED: May-07-2024.

[2] Coffman, E. G., and Graham, R. L. Optimal scheduling for two-processor systems. *Acta Informatica 1*, 3 (Sept. 1972), 200–213. ACCESSED: May-10-2024.

[3] Dodaro, C., and Maratea, M. Nurse Scheduling via Answer Set Programming. 301–307. Series Title: Lecture Notes in Computer Science. ACCESSED: May-07-2024.

[4] Dvision of Information Technology – Stony Brook University. Schedule Builder. https://it.stonybrook.edu/services/schedule-builder, 2016. ACCESSED: Mar-11-2024.

[5] Gebser, Martin and Kaminski, Roland and Kaufmann, Benjamin and Schaub, Torsten. Multi-shot ASP solving with clingo. https://www.cs.uni-potsdam.de/wv/publications/DBLP_journals/corr/GebserKKS17.pdf, 2017. ACCESSED: Apr-21-2024.

[6] pandas development team, T. pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.3509134, Feb. 2020. ACCESSED: Apr-21-2024.

[7] Tange, Ole. *GNU Parallel 2018*. Ole Tange, Mar. 2018. ACCESSED: May-10-2024.

7 Appendix

7.1 Changelog (Change Log)

The most recent/changes updates are at the top.

7.1.1 May 10, 2024

- Removed preferred course selection option.
- Only focused on BS degree, now disregarding MS and PhD degrees.
- Updated tools used in project (added BeautifulSoup4).
- Removed support for ErgoAI (some functions are still available however).
- Included flow charts to describe system design.
- Included updated clingo code for 2 semester scheduling (for fall and spring).
- Added the results of manual testing and performance evaluation.
- Update acknowledgements section.

7.1.2 April 26, 2024

- Removed/exchanged these tools and features:
  - Large Language Models (LLMs) were removed.
  - Sub-graph extraction (via Neural State Machine for Knowledge Base Question Answering) was exchanged for just scraping Stony Brook's SOLAR course catalog for knowledge graph creation.
  - Automated rule creation from knowledge base was removed (outside the scope of this project).
  - Course reviews feature was removed.
  - ErgoAI was exchanged for Clingo.
- Design Document
  - Linked a Google Document with the relevant information and details.
  - Added UML diagrams to pictorially describe the project's code base.
  - Added more explicit design and implementation details.
  - Updated third party libraries and external dependencies used.