1a Course Scheduler

Team 3

Alyssa Chvasta
Vanderbilt University
Computer Science
Nashville, TN
alyssa.j.chvasta

Chris Glenn
Vanderbilt University
Computer Science
Nashville, TN
christopher.a.glenn

Daiwei Lu
Vanderbilt University
Computer Science
Nashville, TN
daiwei.lu

Kamala Varma
Vanderbilt University
Computer Science
Nashville, TN
kamala.m.varma

***Abstract*—We built a recursive, backtracking course planner. We employ a two-stage solution with a course finder and a schedule planner. We then implemented course macros and various improvements to handle requirements intricacies not explicitly encoded by the course dictionary. A performance increase was observed when with the presence of the improvements. We solved the elective satisfaction problem with a unique general solution.**

***Index Terms*—planner, recursion, dfs, macros**

I. Architectural Overview

The problem of the course scheduler can be broken down into two major components: finding a satisfying list of courses and then arranging this list of courses into an eight semester schedule. The first element of the course scheduler is creating a list of satisfying courses using Greedy Depth First Search on an implicit Directed Acyclic Graph. We chose to use this strategy for efficiency, and DFS is beneficial because of the extremely large branching factor while the greedy aspect best handles the large number of possible satisfying schedules by reducing the space searched. The scheduler has two interesting features: the usage of macros representing pre-computed sequences of courses to satisfy an abstract high level goal and a specialized data structure representing goals and what courses satisfy them. The goals data structure is employed to prevent the problem of one course counting for multiple electives.

Although the existence of a satisfying list of courses does not guarantee that an eight semester schedule could be made, or alternatively a course list that does not fit in eight semesters does not guarantee that no schedule exists, we have found that for our purposes it is sufficient to assume this. The current code could be easily expanded to continue searching for satisfying course lists until are possible lists are found and rejected or a course list is found that satisfies the eight semester requirement. Using this assumption we were able to use an abstract schedule class which, along with the course list function, could be developed independently and in parallel.

The schedule class is used to create a valid schedule given a list of satisfying courses. This allows us to abstract the specific scheduling concerns, ie number of hours, pre-requisites, and which semester courses are offered, away from the course scheduler which simplifies it greatly. The scheduler puts each class in the first semester freshman year then checks its and the other class' prerequisites to determine which, if any, courses need to be moved to a later semester to satisfy prerequisite requirements. This optimizes the schedule for early graduation and fewer hours senior year. The schedule class also automatically fills semesters to the 12 hour minimum by selecting random classes unless the semester is Senior Spring which is allowed to be underloaded. In the current implementation of the code, if no schedule can be created in eight semesters with less than or equal to eighteen hours a semester an error is thrown and the program exits.

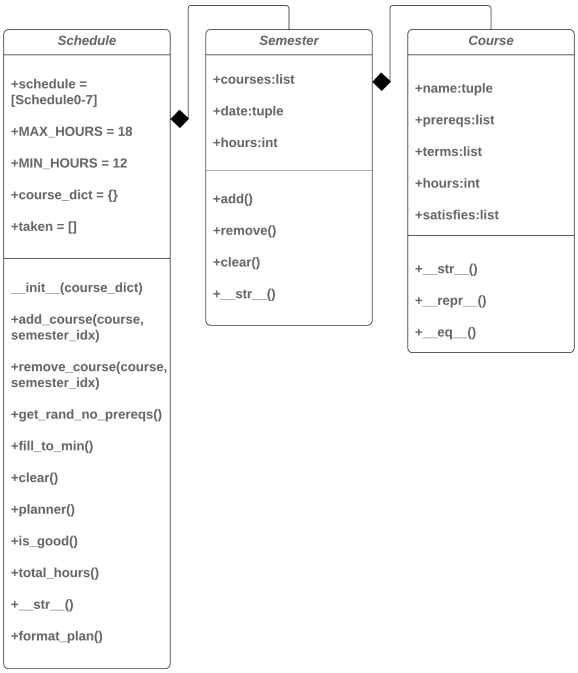

Fig. 1. UML Diagram of the object classes created for our architecture.

II. Design Choices

A number of issues were encountered during the implementation which we will discuss in this section. The decision to create a two-stage scheduler as opposed to one stage was one of intentions. We argue that it is more important to know fully what courses are required to be taken for certain requirements

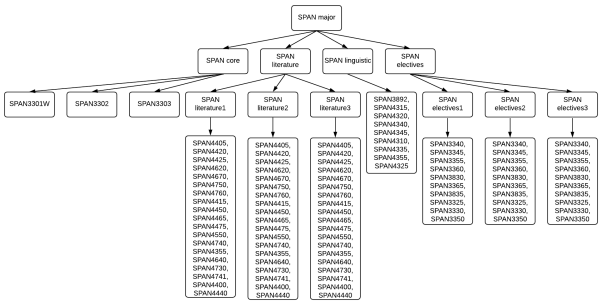

Fig. 2. A major we created, represented as a directed acyclic graph,to provide a simplified test of our scheduler.

than to have a partial list. Thus, requirements to satisfy a goal state will always be known. Prioritizing courses in different orders during the second step led to very interesting results as well. We chose to always push up the first added course of a semester during scheduling to make room as it led to much more natural course progressions. Pushing the last added course often resulted in 1 or 2 less required courses but resulted in unnatural orderings (e.g. ES 140x courses taken during Junior year) not expected by our greedy method.

A core issue discussed during development was how to handle open elective scheduling as it conflicted with standard double counting conventions. We found a general solution to this in our first stage by tracking what requirements each course we scheduled was for. If a course was scheduled for a high-level 0-hour requirement, it could not be scheduled for another within the same degree of study. Thus, taking CS major and SPAN major will result in shared electives but not within the majors themselves.

III. Evaluation

A suite of tests were developed to help with evaluation of the developed scheduler. To complement these tests and allow for simpler development, we developed a Spanish major which was added and to complement the existing Computer Science major. This new major's structure was illustrated in Fig. 2 and provides similar challenge cases as within the Computer Science major. There is the presence of elective credits and multiple courses for a single requirement.

The tests were all passed running on the following equipment.

OS: macOS High Sierra 10.13.4

CPU: 1.6 GHz Intel Core i5-5250U

RAM: 8 GB 1600 MHz DDR3

A. Trivial Cases

Impossible goal: For this test, we called the course scheduler on a goal of all of the ECON and EDUC courses. There are over 100 total, which is impossible to satisfy in 4 years, so we expected an empty plan to be returned.

No goal: The course scheduler was called with an empty goal and should return an empty plan.

TABLE I
Trivial Tests

Test	Execution Times	
	CPU	***Wall***
Impossible goal	0.0s	0.0s
No goal	0.000032s	0.000034s
Already Satisfied goal	0.000053s	0.000053s

Already satisfied goal: In this case, we called the course scheduler with a goal condition that was included in the initial state, which should return an empty plan because the goal has already been satisfied.

B. Simple Cases

TABLE II
Simple Tests

Test	Execution Times	
	CPU	***Wall***
One goal	0.000173s	0.0175s
Initial State	0.000143s	0.000152s
Simple Plan	0.017637s	0.017686s
Simple Plan w/o Macros	0.018176s	0.018209s

One goal: This test is for a scenario where only one goal is provided. This particular goal and its requirements would not themselves completely fill the semesters that are scheduled, so this test is making sure that the semesters end up being filled to reach the minimum number of hours.

Initial state: This test is simply testing a call to the course scheduler with a nonempty initial state. The test is making sure that the course in the initial state is not included in the final plan.

Simple plan: Here we are testing a relatively simple plan. The scheduler was prompted with the goal of MATH2410, which requires some strategic planning due to all of the prerequisites that are required, but it results in a plan that is small and easy to test.

Same simple plan without macros: We repeated the previous test without using macros so that we could compare the execution times. The fact that this test's execution was slower proves that the macros helped speed up the scheduler. It was sped up by a very small amount of time, but this is a very small example.

C. Edge Cases

TABLE III
Logistics Tests

Test	Execution Times	
	CPU	***Wall***
5 credits	0.000178s	0.000190s
Open electives	0.030589s	0.030820s
Proper Terms	0.001576s	0.001625s
CS major logistics	0.010649s	0.010736s

5 credits: Here we are calling the scheduler with a goal of 4 5-credit courses. This makes sure that our scheduler will make plans within the 12-18 hour restriction, even in cases where 5-credit courses cause the total hours to not exactly match up to 12 or 18.

Open elective issue: This is testing the issue that Group 4 brought up. All electives need to be different even though a single elective can be considered satisfactory for each open elective on their own. This test simply checks that when the goal is ('CS', 'openelectives'), the resulting plan has no duplicate courses, but it still has a course to satisfy each of the 6 open elective requirements.

Proper terms: In this test, the scheduler is prompted with a goal that should be completed in only 5 semesters. This test makes sure that not all semesters are scheduled and that the ones that are scheduled are in the proper terms (i.e. the first 5 terms).

CS major logistics: Instead of testing every course and requirement in the CS major since it is complex and ambiguous in a lot of places, we test all of the logistics (proper number of hours, proper terms, etc.).

IV. Alternative Approaches

We consider a few alternatives planner architectures inspired by methods from within the AI field.

A. Within Search

A multitude of search strategies have been developed in the AI field. Many of these strategies were developed in heuristic functions in mind to guide a planner to its goal.DFS has been discussed at length in presentations and talks given by other groups. BFS has also been cast aside as infeasible due to most goal states residing at the bottom of the search tree. Extensions of these two basic search methods were considered for this project and we will discuss two.

A* search and its variants (D* Lite and Focussed D*) are some of the most widely used search algorithms in real world applications [1] [2] [3]. It is a version of best first search (BeFS) modified with a heuristic function. This search algorithm is relatively straightforward to implement but as with most algorithms we will discuss, it requires a consistent heuristic function.

Tabu search was another discussed search algorithm which has received significant attention in its lifetime [4]. Tabu search has an interesting application to this problem as it offers the ability to follow rules defined by the user. It uses memory to store recently visited areas which offered no improvement and avoids them in the near-future, skipping over many unnecessary nodes of a search tree. For the course scheduler, rules can be used to constrain which courses Tabu search might visit, whilst still allowing for experimentation and occasional exploration as opposed to exploitation.

B. As Optimization

The course scheduler can be also be viewed as an optimization problem. One way of posing this problem is that of minimization of number of courses to be taken to satisfy the goal requirements. For example, an action in this state space will still be to add a course to the list of courses taken. It is not guaranteed to make any progress towards the solution. However, you know that if you were to stay at Vanderbilt and take every single course available, you would surely have satisfied a Computer Science major. The optimization problem is to minimize this number of courses which would likely be reducing courses somewhere from 2500 taken to 50. A central issue with utilizing optimization, as will be discussed through attempting applications of different methods will reveal, is defining a way to treat the scheduling problem in some n-dimensional space which can be and formulating a numerical method of traversing this space.

A litany of methods for optimization exist with different strategies in both. A very widespread and basic example is Nelder-Mead Optimization, which is also known as the Simplex Method [5]. It involves moving a simplex or polytope, which is any shape with lines connecting n+1 vertices in any n dimensional shape. An example being a triangle in 2-dimensions or a line in one-dimension. At each step, the vertex with the highest value according to an evaluation function is lowered. An issue with applying this method though, is the need to find some conversion of any given schedule into a space which the vertices can occupy. In this regard, if each vertex were to represent a schedule, the vertex which evaluates to the farthest from desired would be moved elsewhere (finding a different schedule or courselist). At simplest this could be divided into a one-dimensional problem where the dimension is the length of the courselist.

C. Towards a Heuristic Function

Following more recent trends, neural networks could be applied to this problem. Following a method of flattening into relational descriptors and values as in the TRESTLE system by Maclellan et. al [6], we could create an encoding of our schedule which can be converted into a function of "goodness". A neural network could then be used to perform regression on the encoding and predict its "goodness" based on sample schedules we have annotated. This method could be used as an evaluation function with which methods from optimization or search can be applied. Monte Carlo Tree Search has been used with neural networks to great effect in recent literature [7] [8].

V. Extending the Scheduler

A. Benchmarking

Currently we use cpu and wall clock times. Further extensions would look at nodes expanded during the algorithmic search and classes pushed up during the schedule planning phases.

B. Macros and Variabilization

We implemented macro classes to prefilter classes before searching. Instead of performing a full depth first search, for each high-level requirement we specify, we perform a

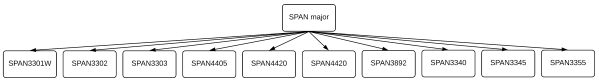

Fig. 3. Macro version of the Spanish Major.

local search and treat the results as a single unit. We then provide these assembled macros to the algorithm and reduce the depth of search space required to 1 or 2 layers at most. In implementation this is done on the top level requirements for the Computer Science major, resulting in a significant speedup. This implementation results in all course prerequisites being collapsed into the second layer with no need for further prerequisite checking. This lack of further chained prerequisite checking results in the speedup observed.

In Fig. 3, the macro version of the Spanish major can be seen where all of the courses have been expanded. Not included in this list are the prereqs for each course which would be included and added without the need for extra checking. They are grouped into dictionaries with requirements from level 2 of Fig. 2 as keys.

Our implementation of variables (Fig. 1 means that courses can be scheduled as entity groups and we can track precisely what course is being used to satisfy what requirements. Converting everything into dynamic variables has resulted in a minor slow-down in our solution as well as memory bloat. However, tracking relationships as they are occurring through recursion and the ability to create new relationships is a marked benefit. Slowdowns are negligible as our solution is fairly greedy. It also opens the way for many future extensions. Creating new courses to be added is simple and is also capable of expressing non-credit supercourses. Beyond our implementation, it is possible to create an interface where any one or group of classes can be easily found and replaced by a user. In our CS-Major, it would be easy to figure out what requirements MATH 1200/1201/1301/2300 satisfy and allow a user to choose a different course to take their place. Similarly, in the Spanish major, our variables allow us to quickly find alternatives if a user chose to replace an elective.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. Koenig and M. Likhachev, "Dˆ* lite," *Aaai/iaai*, vol. 15, 2002.

[3] A. Stentz *et al.*, "The focussed dˆ* algorithm for real-time replanning," in *IJCAI*, vol. 95, 1995, pp. 1652–1659.

[4] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[5] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[6] C. J. MacLellan, E. Harpstead, V. Aleven, and K. R. Koedinger, "Trestle: Incremental learning in structured domains using partial matching and categorization," in *Proceedings of the 3rd Annual Conference on Advances in Cognitive Systems-ACS*, 2015.

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

VI. APPENDIX A

Input: Goal: [('CS', 'major')], Initial State: []
Schedule found in 0.004999 wall seconds.
Schedule found in 0.005640 cpu seconds.

('Fall', 'Frosh')
Courses[('FLUT', '1000'), ('HIST', '2700'), ('ES', '1403'), ('ES', '1402'), ('ES', '1401'), ('CHEM', '1601'), ('MATH', '1300'), ('MATH', '1200')]
Hours: 17

('Spring', 'Frosh')
Courses[('AADS', '3104W'), ('BSCI', '1100'), ('PSY', '1200'), ('MATH', '1201')]
Hours: 12

('Fall', 'Soph')
Courses[('CLLO', '1000'), ('CS', '1101'), ('ARTS', '1102'), ('ENGL', '3896'), ('MATH', '1301')]
Hours: 14

('Spring', 'Soph')
Courses[('COMP', '1000'), ('BSCI', '1511'), ('BSCI', '1100L'), ('EECE', '2116L'), ('EECE', '2116'), ('MATH', '2300')]
Hours: 12

('Fall', 'Junior')
Courses[('MATH', '2400'), ('CS', '2201'), ('CS', '2212'), ('MATH', '2420'), ('MATH', '2410')]
Hours: 16

('Spring', 'Junior')
Courses[('CLAR', '1000'), ('CS', '3250'), ('CS', '2231'), ('CS', '3251'), ('MATH', '2810')]
Hours: 13

('Fall', 'Senior')
Courses[('CS', '3259'), ('CS', '3860'), ('CS', '3281'), ('CS', '3270'), ('SOC', '3702'), ('BSCI', '1510')]
Hours: 18

('Spring', 'Senior')
Courses[('BSSN', '1000'), ('BASS', '1000'), ('CS', '1151'), ('CS', '1103'), ('PHYS', '1601'), ('CS', '3861')]
Hours: 14

Total Hours: 116

Input: Goal: [(CS, Major)], Initial State: [(‘CS’, ‘1101’),(‘MATH’, ‘1200’)]
Schedule found in 0.005000 wall seconds.
Schedule found in 0.004781 cpu seconds.

('Fall', 'Frosh')
Courses[('FLUT', '1000'), ('HIST', '2700'), ('ES', '1403'), ('ES', '1402'), ('ES', '1401'), ('CHEM', '1601'), ('MATH', '1300'), ('MATH', '1201')]
Hours: 17

('Spring', 'Frosh')
Courses[('AADS', '3104W'), ('BSCI', '1100'), ('PSY', '1200'), ('MATH', '1301')]
Hours: 13

('Fall', 'Soph')
Courses[('CLLO', '1000'), ('CS', '2201'), ('ARTS', '1102'), ('ENGL', '3896'), ('MATH', '2300')]
Hours: 13

('Spring', 'Soph')
Courses[('COMP', '1000'), ('BSCI', '1511'), ('BSCI', '1100L'), ('MATH', '2810'), ('EECE', '2116L'), ('EECE', '2116'), ('MATH', '2420'), ('MATH', '2410')]
Hours: 18

('Fall', 'Junior')
Courses[('CS', '1103'), ('SOC', '3702'), ('BSCI', '1510'), ('CS', '2231'), ('CS', '3251'), ('CS', '2212')]
Hours: 18

('Spring', 'Junior')
Courses[('CLAR', '1000'), ('CS', '3861'), ('CS', '3860'), ('CS', '3281'), ('CS', '3270'), ('CS', '3250')]
Hours: 16

('Fall', 'Senior')
Courses[('CS', '4959'), ('ARTS', '2101'), ('BSSN', '1000'), ('BASS', '1000'), ('CS', '1151'), ('MATH', '2400'), ('PHYS', '1601')]
Hours: 16

('Spring', 'Senior') Courses[] Hours: 0 Total Hours: 111

Input: Goal: [(‘CS’, ‘3250’)], Initial State: []
Schedule found in 0.010999 wall seconds.
Schedule found in 0.011300 cpu seconds.

('Fall', 'Frosh')
Courses[('CS', '2212'), ('CS', '1101'), ('ENGL', '3662'), ('HOD', '1250')]
Hours: 12

('Spring', 'Frosh')
Courses[('CS', '2201'), ('EES', '4650'), ('SOC', '3312'), ('LPO', '3936')]
Hours: 12

('Fall', 'Soph')
Courses[('CS', '3250'), ('NURS', '1601'), ('CMST', '1500'), ('HIST', '1190'), ('CLLO', '1100')]
Hours: 14

('Spring', 'Soph')
Courses[]
Hours: 0

('Fall', 'Junior')
Courses[]
Hours: 0

('Spring', 'Junior')
Courses[]
Hours: 0

('Fall', 'Senior')
Courses[]
Hours: 0

('Spring', 'Senior')
Courses[]
Hours: 0
Total Hours: 38

Input: Goal: (A conjunction of all ECON and EDU classes)
Schedule found in 0.720010 wall seconds.
Schedule found in 0.720427 cpu seconds.

('Fall', 'Frosh')
Courses[]
Hours: 0

('Spring', 'Frosh')
Courses[]
Hours: 0

('Fall', 'Soph')
Courses[]
Hours: 0

('Spring', 'Soph')

Courses[]
Hours: 0

('Fall', 'Junior')
Courses[]
Hours: 0

('Spring', 'Junior')
Courses[]
Hours: 0

('Fall', 'Senior')
Courses[]
Hours: 0

('Spring', 'Senior')
Courses[]
Hours: 0
Total Hours: 0
