Redesigning WebTree

George Baldini and **Michael Robertson**
{gebaldini,mirobertson}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

We model Davidson College's problem of class assignment as an integer programming constraint satisfaction problem using anonymized data collected from 2013-2015 student WebTree submissions. Our results provide a legitimate argument for replacing WebTree. Our best algorithm outperforms WebTree more often in our tests and also incorporates more factors into its decision-making. Our solution improves the percentage of students receiving classes in their major but violates WebTree logic, ignoring class dependencies (i.e. "I want this class only if I don't get that class"). We conclude with ideas to further improve our algorithm.

1 Introduction

The Davidson College Registrar manages course assignments from over 400 courses to each of Davidson's $\approx 2,000$ students each semester. To handle this problem, the Registrar uses a front-end application called "WebTree" to record student preferences and back-end COBOL code, written by a retired professor, to distribute classes based on preferences.

Davidson students, particularly upperclassmen, frequently criticize WebTree because it often leaves them without classes they need to graduate. In this paper, we offer an alternative course selection algorithm that models this problem as a linear programming constraint optimization problem. We constructed our model with WebTree input data from the fall and spring semesters of the 2013 and 2014 academic years. Each file contained the following columns:

- ID: student ID number
- CLASS: class standing of student
- CRN: unique identifier for a specific section of a course (what students select on WebTree)
- TREE: the tree number in which the current CRN was listed (1, 2, 3, or 4)
- BRANCH: the node in the tree where the CRN was listed. Nodes are numbered in level order, left-to-right, starting at 1
- COURSE_CEILING: enrollment limit for this CRN
- MAJOR: student's major
- MAJOR2: student's second major
- SUBJ: course subject code
- NUMB: course catalog number
- SEQ: course section number

In section 2 we describe the process WebTree currently goes through in order to assign classes, and in section 3 we describe our own approach to modeling the problem. We present our results in section 4 and offer suggestions for improvements to our own model in 5.

2 Background

The WebTree front-end web application allows students to imput their preferences to 25 nodes on 4 trees. Each node represents a course preference logically dependent on other potential assignments. Figure 1 describes the logical dependency of these nodes.

We modeled the WebTree input data as a ranked list of 48 choices, where each choice is a set of four classes. For example, a student's first choice for a set of four classes would be their 1 node class, their 1A node class, their 1AA node class, and their 4A node class. Their 48th choice for a set of four classes would be their 3 node class, their 3B class, their 3BB class, and their 4D class. WebTree also includes a "second pass" that executes if a student has less than four classes after exhausting all 48 choices (as referenced in the 4D node in Figure 1). However, since satisfactory explanation has been provided by relevant parties, we have excluded it from our interpretation of WebTree's functioning.

The WebTree back-end algorithm first groups students based on seniority, and then randomly ranks them within each class, assigning them a lottery number. Moving through the lottery numbers sequentially, it assigns a student their top class choice if the assignment meets the following requirements:

- The class has not exceeded its capacity.
- The student has not already been assigned this class.
- The student has not already been assigned four classes.

In the case that a student's top choice violates one of these conditions, WebTree looks to their next choice, and continues on until it either assigns a student a course or exhausts their preferences.

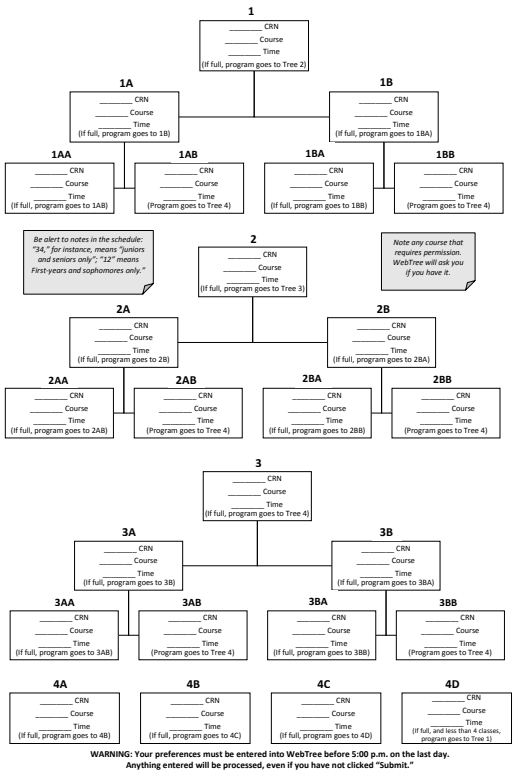

Figure 1: WebTree Logic. Image courtesy of the Davidson College Registrar (Davidson College).

3 Experiments

In our solution of the problem, we modeled the final assignments in a student-class matrix with Boolean entries, which lent itself to a mathematical description of the relevant constraints and quality of a solution.

For the variables of our CSP, we used a matrix of size S by C, where S is the total number of students who submitted WebTree preferences, and C is the total number of classes offered. Each entry in the matrix P_{ij} therefore corresponded to a specific student-class combination. Each entry took a Boolean value, where of 0 corresponded to that student not taking the class, and 1 corresponded to the student taking the class.

Our constraints were:

1. Every student must have between 2 and 4 classes:

$$2 \leq \sum_{j=1}^{C} P_{ij} \leq 2 \quad \forall \text{ rows } i$$

2. Every class must have a number of students in it between 0 and its capacity:

$$0 \leq \sum_{i=1}^{S} P_{ij} \leq cp(C_j) \quad \forall \text{ columns } j,$$

   where $cp(C_j)$ is the capacity of the class corresponding to column j.

3. No student can be in a class more than once. (Implicitly enforced through the modeling of the problem)

Solving the CSP requires an optimization function to describe quantitatively how well-suited a class assignment is to a student's choices. In our experiments, we varied this function considerably to improve our results, but consider as an example our original choice:

$$f_1(S_i, CRN) = g_i(CRN) \cdot h(CRN, g_i(CRN)) \cdot y_1(S_i) \cdot m_1(S_i, CRN)$$

where

- The function $g_i(CRN)$ is 48 minus number of the ith student's choice in which the CRN is found. For example, if course CRN 25143 is first found in a student's 36th ranked WebTree choice of four classes, $g_i(25143) = 48-36 = 12$.
- The function $h(CRN, g_i(CRN))$ is the location of a class CRN within the ith student's ranked choice $g(CRN)$. For example, if course CRN 25143 is found in a student's 36th ranked WebTree choice of four classes, and it is the 3rd choice class in that choice of four classes, $h(CRN, g(CRN)) = 3$.
- The function $y_1(S_i)$ is the student's class year (senior = 4, ... etc).
- The function $m_1(S_i, CRN)$ returns 2 if the CRN references a class in student's S_i's major, and 1 otherwise.

By supplying Google's "Glop" integer programming solver with different versions of f as our optimization function, we used them to find the optimal constrained assignment of students to classes (Perron 2011).

We created several different metrics of success to measure our solution against the original WebTree algorithm and modified our optimization function f in order to improve our results.

We first ran Glop using the example optimization function f_1, and then modified f_1 after our solution did not outperform WebTree on our metrics. We remapped the output to change the relative weights between choices. Because g_i and h_i in f_1 are both linear functions of the class-student pairing's position in a list, a change by k positions in either of these lists corresponds directly to a k change in the output of that function.

This change doesn't reflect all of the students' actual preferences however. Students generally care much more about moving from their fifth WebTree choice of four classes to their sixth choice than about moving from their twenty-seventh choice to their twenty-eighth. To reflect this, we mapped the output of g and h using two functions, a polynomial and a sigmoid, which preserved the ranges on the domains, but altered the rates of change of the optimization value at different points, corresponding to students actual preferences. We also decreased the value of giving upperclass students their first choice, as well as giving majors their major choices. We wrote $y_2(S_i)$ to be such that $y_2(1) = 1.2, y_2(2) = 1.5, y_2(3) = 1.8$, and $y_2(4) = 2$ (where 1 corresponds to a first year student ...etc), and wrote $m_2(S_i, CRN)$ to return 1.3 if the class matched the students major, rather than 2 as m_1 did. Both return 1 if the class and major do not match.

We first used a polynomial generated from interpolation through the points (48,48) (1,1) (20,10) to map the output. The first two points specified that the range doesn't change, and the third point seems like a reasonable assumption: the 20th choice should be worth approximately a fifth as much as the first choice. Lagrange polynomial interpolation through these points produced the mapping function

$$P(x) = \frac{5}{266}x^2 - \frac{3}{38}x + \frac{20}{10}$$

Using this remap to modify the outputs of g_i and h, we create

$$f_2(S_i, CRN) = P(g_i(CRN)) \cdot P(h(CRN, g_i(CRN))) \cdot y_2(S_i) \cdot m_2(S_i, CRN)$$

We intended this mapping function to model the fast drop-poff in valuation of choices. It attempts to account for the fact that students care much more about changing from their first choice to their third choice than from changing from their twentieth choice to their twenty-first choice. However, because of the drastically poor performance of the optimization on measures of success described in our results, we reworked our mapping function.

We created f_3 by changing our modeling function from an interpolating polynomial to a sigmoid. This function models the high valuation by students of their top choices, and the sharp drop-off in valuation around their sixth choice. Using a new sigmoid mapping function

$$S(x) = \frac{48}{1 + e^{-0.5(x-37)}}$$

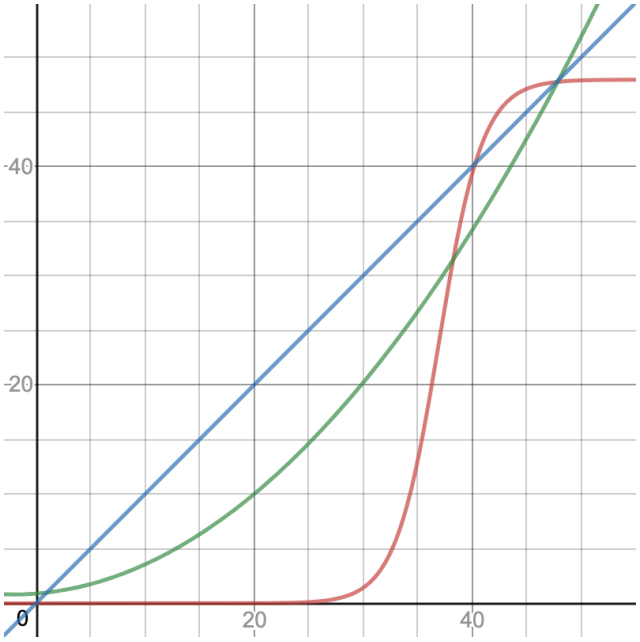

Figure 2: f_1 (Blue), f_2 (Green), and f_3 (Red). Each function attempts to model student valuation of relative parts of their WebTree rankings. Image courtesy of Desmos Grapher.

we again modified the outputs of g_i and h to create

$$f_3(S_i, CRN) = S(g_i(CRN)) \cdot S(h(CRN, g_i(CRN))) \cdot y_2(S_i) \cdot m_2(S_i, CRN)$$

We compare the results of these different optimization functions in the next sections.

4 Results

As shown in Figure 3, each algorithm had varying results. Our f_3 gave a higher percentage of students their first and second choices for most semesters, while WebTree (f_0) performed better for students' third and fourth choices. We do not know why f_2 performed so miserably, but we can conclude that a polynomial mapping function does not correctly model students' interests.

Our integer programming solver with f_1 and f_3 beat WebTree in assigning more major classes to students who asked for them. We expected this outcome, because we added a positive factor in our optimization function based on whether class-student pairing matched a student with a class in their major.

We have bolded the highest percentage in each row for each year. In total f_3 has 19 bolded entries, f_2 has 0, f_1 has 8, and f_0 has 14. This suggests that our linear programming model better solved the WebTree model than the baseline current application of WebTree, but only when using the f_3 optimization function.

5 Conclusions

In this paper, we sought to improve Davidson's WebTree algorithm by modeling it as constraint satisfaction problem.

Category	Fall 2013				Spring 2014				Fall 2014				Spring 2015			
	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3
1st Choices	88	82	22	**90**	90	81	25	**92**	91	83	21	**93**	91	84	25	**92**
2nd Choices	75	76	22	**81**	67	69	23	**71**	74	75	23	**79**	73	73	22	**77**
3rd Choices	58	**59**	22	54	50	**54**	23	49	**56**	53	21	47	**58**	57	21	51
4th Choices	**58**	36	23	29	**55**	38	22	30	**58**	34	23	25	**62**	38	22	30
Major 1st Choice	88	**94**	24	**94**	91	94	26	**96**	94	**97**	21	**97**	90	94	27	**95**
Major 2nd Choice	77	**90**	24	85	69	**84**	24	76	76	**91**	25	84	75	**85**	24	81
1st Choices Seniors	**94**	85	25	91	**96**	89	31	94	**95**	89	26	95	**98**	95	31	**98**
1st Choices Juniors	87	86	25	**92**	92	85	24	**94**	**97**	89	22	**97**	94	88	25	**95**
1st Choices Sophomores	81	72	20	**83**	85	77	22	**86**	88	75	20	**89**	**87**	77	22	**87**
1st Choices Freshmen	74	70	15	**77**	83	70	21	**87**	87	81	16	**90**	86	76	20	**89**

Figure 3: Comparison of our algorithms (f_1 - linear, f_2 - polynomial, f_3 - sigmoid) to the traditional WebTree, f_0. An entry in "x" Choices for function f_i indicates the percentage of students who received their xth choice, where their first choice is their 1 node class on WebTree, their second is their 1A class, third is their 1AA class, and fourth is their 4A class. Major First Choice and Second Choice refer to the percentage of students whose first and second choice, respectively, were classes in their major, and who got that class. First choice percentages are also broken down by year in the bottom four columns.

Our results offer an argument to replace WebTree with a linear programming solver using the f_3 optimization function, as our algorithm produces the highest percentages most often in Figure 3. Additionally, our algorithm also considers an additional factor in this problem: students' majors. Due to the common dissatisfaction of students not receiving classes in their majors, we provide the functionality to remedy that inconsistency.

In order for our algorithm to maximize its payoff, it would require replacing the data input portion of the current WebTree program. Our approach can offer students a set of classes that they didn't request together, so we suggest modifying the front-end data input part of WebTree. We would solicit a one "must-have" course, two sets of three courses of which the student would highly value getting one of, and a remaining set of six courses the student would like one of, but prioritizes least. The current WebTree data intake focuses heavily on the logical dependency of often redundant class dependencies, forcing the student to describe their preferences in repetitive detail. Because students generally want one to two courses for their major, one or two for a minor or distribution requirement, and one for general interest, students' choices generally don't have complex dependencies, making much of the logical structure of WebTree useless and students' entries in it redundant. For example, while a student may only want to take one 300-level anthropology class out of a ranked list of three, the course assignment they receive from that group is unlikely to effect their their ranked list of three language courses they'd like to use to fulfill their language requirement.

While our approach rarely makes gains of more than 10% on our metrics over the baseline WebTree, our model does allow for easy adjustment of class year weighting. If, for example, seniors successfully convinced the administration through student government that their preferences should be weighed more heavily over first-years than our model currently does, we could easily accommodate this request by simply changing a weight in our optimization function, while the baseline WebTree has no similar easily implementable fix. Similarly, if large numbers of students do not receive the classes in their major they need to graduate, our WebTree algorithm could simply be rerun with a larger output on our m function in the case of a match, and the new student-class assignment solution produced by rerunning Glop would accommodate the desired change.

6 Contributions

Baldini and Robertson both contributed to each function of the code. Robertson focused more on creating our model, while Baldini spent more time working on the data intake and testing cases. Both authors co-authored each section of the paper and proof-read the entire document.

7 Acknowledgements

Thank you Dr. Ramanujan for guidance throughout the process of implementing our algorithm.

References

Davidson College. WebTree Worksheet. `https://www.davidson.edu/Documents/Administrative%20Department/Registrar/WebTreeWorksheet.pdf`. Retrieved on Mar. 25, 2019.

Perron, L. 2011. Operations Research and Constraint Programming at Google. In Lee, J., ed., *Principles and Practice of Constraint Programming – CP 2011*, 2–2. Berlin, Heidelberg: Springer Berlin Heidelberg.