An Algorithmic Approach to Address Course Enrollment Challenges

Arpita Biswas
Harvard University, Cambridge, MA, USA

Yiduo Ke
Northwestern University, Evanston, IL, USA

Samir Khuller
Northwestern University, Evanston, IL, USA

Quanquan C. Liu
Northwestern University, Evanston, IL, USA

Abstract

Massive surges of enrollments in courses have led to a crisis in several computer science departments - not only is the demand for certain courses extremely high from majors, but the demand from non-majors is also very high. Much of the time, this leads to significant frustration on the part of the students, and getting seats in desired courses is a rather ad-hoc process. One approach is to first collect information from students about which courses they want to take and to develop optimization models for assigning students to available seats in a fair manner. What makes this problem complex is that the courses themselves have time conflicts, and the students have credit caps (an upper bound on the number of courses they would like to enroll in). We model this problem as follows. We have n agents (students), and there are "resources" (these correspond to courses). Each agent is only interested in a subset of the resources (courses of interest), and each resource can only be assigned to a bounded number of agents (available seats). In addition, each resource corresponds to an interval of time, and the objective is to assign non-overlapping resources to agents so as to produce "fair and high utility" schedules.

In this model, we provide a number of results under various settings and objective functions. Specifically, in this paper, we consider the following objective functions: total utility, max-min (Santa Claus objective), and envy-freeness. The total utility objective function maximizes the sum of the utilities of all courses assigned to students. The max-min objective maximizes the minimum utility obtained by any student. Finally, envy-freeness ensures that no student envies another student's allocation. Under these settings and objective functions, we show a number of theoretical results. Specifically, we show that the course allocation under the time conflicts problem is NP-complete but becomes polynomial-time solvable when given only a constant number of students *or* all credits, course lengths, and utilities are uniform. Furthermore, we give a near-linear time algorithm for obtaining a constant 1/2-factor approximation for the general maximizing total utility problem when utility functions are binary. In addition, we show that there exists a near-linear time algorithm that obtains a 1/2-factor approximation on total utility and a 1/4-factor approximation on max-min utility when given uniform credit caps and uniform utilities. For the setting of binary valuations, we show three polynomial time algorithms for 1/2-factor approximation of total utility, envy-freeness up to one item, and a constant factor approximation of the max-min utility value when course lengths are within a constant factor of each other. Finally, we conclude with experimental results that demonstrate that our algorithms yield high-quality results in real-world settings.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases fairness, allocation, matching, algorithms

Digital Object Identifier 10.4230/LIPIcs.FORC.2023.8

Supplementary Material *Software*: `https://github.com/yiduoke/CS-499-Khuller`
archived at `swh:1:dir:d79d76fd785cad2f69c6edf8a15ecac2d81bc363`

4th Symposium on Foundations of Responsible Computing (FORC 2023).
Editor: Kunal Talwar; Article No. 8; pp. 8:1–8:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding Samir Khuller and Yiduo Ke gratefully acknowledge support from NSF-Award 2216970 (IDEAL Institute).

Acknowledgements We thank Prahlad Narasimhan Kasthurirangan for pointing us to a relevant reference for one of the problems we consider.

1 Introduction

This work addresses a central problem in fair resource allocation in the course allocation setting. In the algorithms community, one of the fairness objectives is to allocate resources among agents to maximize the minimum allocation to any single agent, also known as "Santa Claus" problem. In the course allocation setting, there are additional constraints to the Santa Claus problem, such as a "conflict" graph between the resources, in other words, if there is a conflict edge between two resources, then we cannot allocate that pair of resources to the same agent. Our study was motivated by the course allocation scenario since massive surges in enrollments in CS courses have led to a crisis in several computer science departments - not only is the demand for certain courses extremely high from majors, but the demand from non-majors is also very high. Much of the time, this leads to significant frustration on the part of the students who are unable to get into courses of interest, and this lead to non-uniformity in student happiness as a few students were able to successfully petition faculty to add them to their course, and other students failed to get into any course of interest (leading to further annoyance when finding out that you did not get in, but your friend did). As registration opens up, there is always a mad scramble to enroll in courses. Given the amount of money spent by students on fees, and due to the scale of the problem, we set out to collect the information from students about which courses they want to take, and then developed optimization models for assigning students to available seats. What makes this problem complex is that courses themselves have time conflicts, so a student might be interested in two courses, but if they meet at overlapping times, they can only take one of those courses. Moreover, students have credit caps, that limit how many courses a student can enroll in, and naturally, courses have limited capacity. Students specify a set of courses that they are interested in, and we care about total utility (assigned seats), as well as fairness measured by both the lowest allocation to any student in an assignment and envy-freeness.

While our motivating example was assigning seats to students in a fair manner, this is a pretty general resource allocation problem with some additional constraints capturing conflicts among courses and capacity constraints of students. We represent the conflict using a *conflict graph* where resources are the nodes and an edge between two resources implies that those two resources cannot be assigned to the same student.

The problem when the conflict graph is unrestricted is NP-hard (Appendix A). Thus, we focus on the case of assigning resources that can be represented as intervals. Each interval has a start and end time. We assume that time occurs in discrete integer time steps in increments of 1 beginning with step 0. Overlapping intervals are those that strictly overlap (an interval ending at time 3 does not overlap with another interval that starts at 3). The conflict graph is now determined by the overlapping structure: if two resources (intervals) overlap in time, then there is an edge between them in the corresponding conflict graph.

1.1 Related Work

The problem of allocating resources among a set of n agents with an egalitarian objective (maximizing the total value of items allocated to the worst-off agent) has been well-studied in the literature and is known as the Santa Claus problem. This problem was introduced by Bansal and Sviridenko [3] and they developed a $O(\log\log n/\log\log\log n)$ approximation algorithm. Later, Davies et al. [16] improved it to a $(4+\epsilon)$-approximation. More recently, Chiarelli et al. [15] considered the Santa Claus problem assuming conflicting items represented by a conflict graph. They analyzed the NP-hardness of the problem for specific subclasses of conflict graphs and provided pseudo-polynomial solutions for others. Our work complements their results by providing constant approximate (polynomial time) solutions for interval graphs with uniform and binary valuations for course allocation.

Another well-studied fairness criterion in the fair division literature is envy-freeness [17], where every agent values her allocation at least as much as she values any other agent's allocation. However, envy-freeness does not translate well when the items to be allocated are indivisible (for example, if there is one indivisible item and two students, the item can be allocated to only one student, and the other student would envy). Thus, for indivisible items (such as course seats), an appropriate fairness criterion is envy-freeness up to one item (EF1), defined by Budish [12]. Prior works have shown that an EF1 allocation always exists while allocating non-conflicting budgeted courses [12], under submodular valuations [31], under cardinality constraints [7], conflicting courses with monotone submodular valuations and binary marginal gains over the courses [4, 34], and many more. However, these results do not consider interval graphs to model conflicting courses and thus, the existing EF1 solutions cannot solve the fair course allocation problem that we consider. Recent work by Hummel et al. [24] explored the allocation of conflicting items with EF1 fairness criteria. They showed the existence of EF1 for conflict graphs with small components and refuted the existence of EF1 when the maximum degree of the conflict graph is at least as much as the number of agents. Moreover, they provided a polynomial time EF1 solution when the conflict graph consists of disjoint paths and the valuations are binary. Our work extends their results by providing a polynomial time EF1 solution for interval graphs with binary valuations, which are more general than disjoint-path graphs and capture conflicts between courses.

Fair allocation of intervals has been studied in job scheduling problems, where each job is represented as an interval (with a starting time, deadline, and processing time) and is required to be allocated to machines such that the same machine is not scheduled to run another job at the same time. Fairness notions considered are in terms of load balancing [2], waiting time envy freeness [6], completion time balancing [25], and EF1 among machines [30]. However, these papers allow flexible time intervals, which cannot capture conflicts as graph edges and represent a different problem from our work.

Other related techniques to our fair course allocation problem include equitable coloring [8, 27, 20], bounded max coloring [10, 23], mutual exclusion scheduling [18, 26, 33], although most of these works are only tangentially related to our problem at hand. There have also been many works on approximation algorithms for various different types of conflict models [9, 13, 29, 32] and resource constrained scheduling [5] but none of these works operate in the specific conflict graph and allocation model studied in our paper.

1.2 Summary of Contributions

In this paper, we tackle the problem of *fair* allocation of conflicting resources. We prove that a general version of the problem is NP-hard via a reduction from the independent set problem in Appendix A. This motivates the study of a specific class of conflict graphs, namely interval graphs, which capture the course allocation problem. For interval graphs, we provide polynomial time algorithms to obtain a fair allocation. We establish that, oftentimes, *simple* *algorithms are enough to provide multiple guarantees in terms of efficiency and fairness*, specifically, a round robin approach is often sufficient. Figure 1 summarizes our results. Our main results are:

- We first consider *uniform utilities* in Section 3 and show that the course allocation under the time conflicts problem with the objective of maximizing social welfare is NP-complete in general. However, we develop polynomial-time solutions when there are a constant number of students *or* when the credit caps and course lengths are uniform. We further provide solutions that have fairness guarantees, one of which satisfies envy-freeness up to any good (EFX) and the other achieves approximate maxi-min fairness.
- We then investigate *binary utilities and uniform credits for all courses* in Section 4 and develop a (1/2)-approximate solution for the course allocation problem under the time conflicts problem with the objective of maximizing social welfare. We further provide solutions that have fairness guarantees, one of which satisfies envy-freeness up to one good (EF1) and the other achieves approximate max-min fairness.
- Our experimental evaluation demonstrates that our algorithms yield near-optimal solutions on synthetic as well as real-world university datasets.

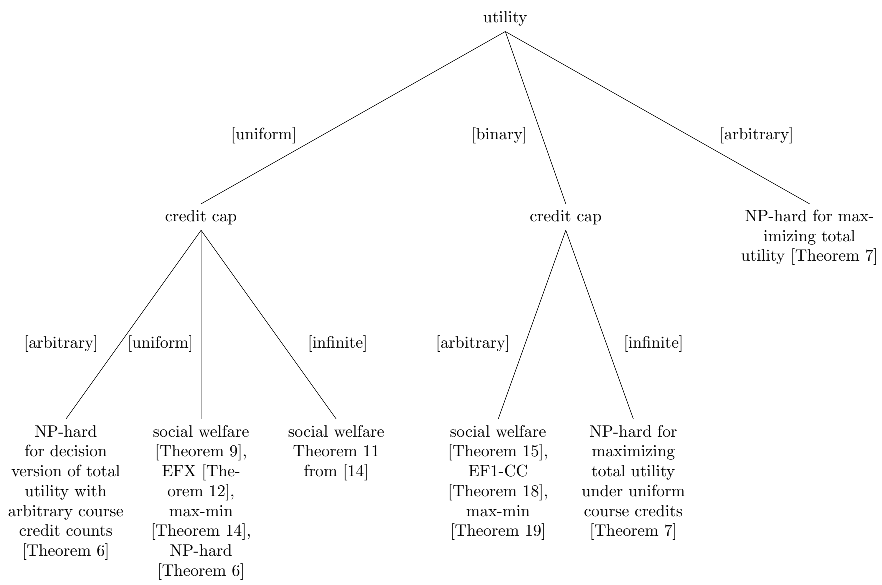

Figure 1 Overview of results.

2 Preliminaries

In this section, we define our problem as well as the necessary concepts for our results. We first define our main problem which we call the Course Allocation Under Time Conflicts problem (CAUTC). This problem describes an issue almost all universities face: given a set of courses that have meeting times during the week and student preferences over these courses, what is the best way to assign these courses to students? Each course has a seating capacity, after all. From a university's perspective, filling seats has value (maximizing utility), but we have to balance that with a fairness aspect as well.

2.1 Course Allocation under Time Conflicts Model

We consider the problem of allocating a set of m courses among a set of n students. Let $\mathcal{N}$ be the set of students and $\mathcal{M}$ be the set of courses. Courses in $\mathcal{M}$ have indices in $\mathcal{M}$. Each student $i \in \mathcal{N}$ has a non-negative utility for each course $j \in \mathcal{M}$; this utility is denoted by $u_i(j) \geq 0$. C_i represents the maximum number of credits, a student i can take. Each course j has a certain number of credits indicated by c_j, a seat capacity of s_j for each $j \in \mathcal{M}$, a start and end time, represented by the tuple $(start_j, end_j)$ and a duration d_j (in units consisting of discrete time steps). Finally, each course j is associated with a seat count s_j. Therefore, the restrictions are:

- A student $i \in \mathcal{N}$ can be matched to courses with the total credits at most C_i (*credit cap*).
- A course $j \in \mathcal{M}$ can be allocated to at most s_j students.
- No student can be allocated a pair of courses that overlap in time.

Although we define the problem in the most general form, for the rest of this paper, we set $c_j = 1$ for all courses. Furthermore, we reduce to the equivalent problem where we make a copy of the course for each seat and create an interval with the same start and end time for each seat of the course. Via this reduction, we also set $s_j = 1$ for all courses.

The course schedule can be represented as an interval graph. We illustrate such a configuration in Figure 2.

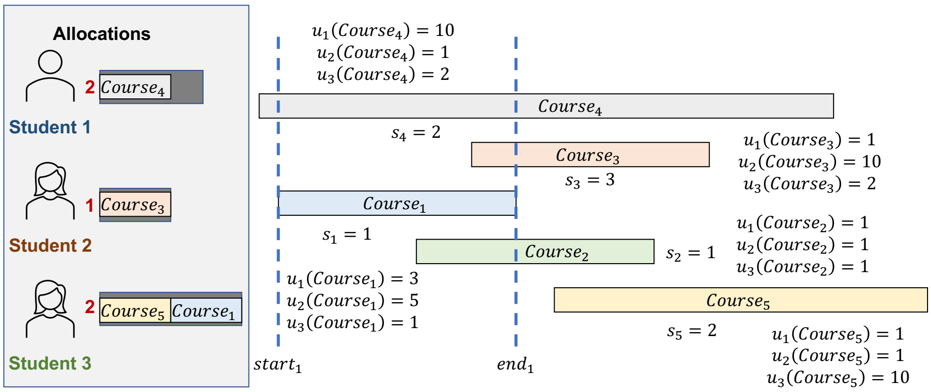

Figure 2 An CAUTC instance with 3 students and 5 courses, with one seat per course. All courses conflict with each other except for Course_1 and Course_5. The red numbers students indicate the credit caps for students. The allocation represents a solution for CAUTC-SW (Definition 1).

2.2 Fairness Measures

We first consider the problem of finding an allocation that maximizes the social welfare (total sum of utilities of all the students based on the courses allocated) subject to all the feasibility and non-conflicting constraints. We call this maximization problem CAUTC-SW.

▶ **Definition 1** (CAUTC-SW). *Given a set of students $\mathcal{N}$, a set of courses $\mathcal{M}$, and the set of utility functions $\mathcal{U}$, CAUTC-SW is the assignment of courses to students such that the social welfare is maximized and the constraints of CAUTC are satisfied.*

In addition to maximizing social welfare, we also consider a number of common fairness measures as constraints. We first define them here but will slightly modify some of these definitions in their respective sections later on in this paper.

We first define the concept of *envy-free up to any good* (EFX). Informally, EFX means that if any agent A were to be envious of any agent B, then A would no longer be envious if any one item were to be removed from agent B's allocation.

▶ **Definition 2** (Envy-Free Up to Any Good (EFX)). *For all students $i \in \mathcal{N}$, if there exists an $i' \in \mathcal{N}$ such that $u_i(A_{i'}) > u_i(A_i)$, then for all items $x \in A_{i'}$, it follows that $u_i(A_{i'} \setminus x) \leq u_i(A_i)$ or $C_i = \sum_{j \in A_i} c_j$ (student i has reached their credit cap), where A_k denotes the allocation of courses to student k.*

A slightly weaker version of EFX is *envy-free up to one good* (EF1), defined below. Informally, EF1 means that if any agent A were to be envious of any agent B, then A would no longer be envious if a particular item were to be removed from agent B's allocation.

▶ **Definition 3** (Envy-Free Up to One Good (EF1)). *For all students $i \in \mathcal{N}$, if there exists an $i' \in \mathcal{N}$ such that $u_i(A_{i'}) > u_i(A_i)$, then there exists an item $a \in A_{i'}$ satisfying $u_i(a) > 0$, such that $u_i(A_{i'} \setminus a) \leq u_i(A_i)$ or $C_i = \sum_{j \in A_i} c_j$ (student i has reached their credit cap), where A_k denotes the allocation of courses to student k.*

The problem with only ensuring EF1 is that there is a trivial allocation of courses consisting of giving everyone one course only or no courses. Such an allocation is EF1 since no one envies anyone else by more than one course. However, such an allocation is not a very useful allocation most students would not receive as many courses as they want and there will be many remaining courses. Thus, we need a better measure of envy. A definition from [30] resolves this problem. Suppose all unassigned courses in each iteration were donated to a dummy student, the ***charity***, who is unable to envy anyone, but students are able to envy the charity. Then, having the charity resolves the issue of trivial solutions. Specifically, any student i can envy the charity by considering the maximum independent set among the courses in the charity that are desired by i. If such a maximum independent set is larger than the number of courses allocated to i, then i envies the charity. We formally define EF1 Considering Charity (EF1-CC) to be our new notion of envy below.

▶ **Definition 4** (Envy-Free Up to One Good Considering Charity (EF1-CC)). *Any student i who has reached their credit cap (i.e. $C_i = \sum_{j \in A_i} c_j$) does not envy anyone else. For all other students $i, i' \in \mathcal{N}$ (who have not reached their credit caps) and given an allocation $\mathcal{A} = (A_1, \dots, A_i, \dots A_n)$ of courses, it holds that $|\{j \mid u_i(j) > 0, j \in A_i\}| \geq |\{j \mid u_i(j) > 0, j \in A_{i'}\}| - 1$. Let D be the set of courses that are unassigned and held by a dummy student defined as the* **charity**. *Let $MIS_i = MIS(\{j \mid u_i(j) > 0, j \in D\})$ be the maximum independent set of courses in D that are desired by student i. Then, for all students $i \in \mathcal{N}$, it holds that $|\{j \mid u_i(j) > 0, j \in A_i\}| \geq |MIS_i| - 1$.*

Finally, we consider a Santa Claus fairness objective which is to maximize the minimum allocation of courses to any student. For simplicity, we denote this problem as CAUTC-SC.

▶ **Definition 5** (CAUTC-SC). *Determine an allocation of courses to students $\mathcal{A} = (A_1, \dots, A_n)$ that maximizes the minimum utility of any student subject to the constraints of CAUTC. Namely, we seek to satisfy the following objective* $\max_{\mathcal{A}} \left(\min_{i \in \mathcal{N}} \left(\sum_{j \in A_i} u_i(j) \right) \right)$.

3 Uniform Utilities for Courses

In this section, we discuss the setting where all students have equal, uniform preferences for all courses. In other words, in this section, all students have preference 1 for every course. In this setting, we show a number of hardness, social welfare, and fairness results described in the following sections.

3.1 Hardness of CAUTC-SW under Uniform Utilities

We show that CAUTC-SW is NP-hard (Theorem 6). Subsequently, we consider some variants of the problem that are polynomial-time solvable in the following sections.

▶ **Theorem 6.** *The CAUTC-SW problem where the utilities are uniform, credit caps are uniform, course are non-overlapping, and number of credits for each course is non-uniform and arbitrary is NP-hard.*

We prove this via a reduction from the 3-partition problem (Appendix B.1).

▶ **Theorem 7.** *The CAUTC-SW problem where utilities are binary, credit caps are infinite, and number of credits for each course is uniform is NP-hard.*

We prove this via a reduction from the k-coloring problem for circular-arc graphs. The complete proof is in Appendix B.2.

3.2 Maximizing Social Welfare

In this section, we show that, for some more restricted settings, the CAUTC-SW problems are polynomial-time solvable. We first show that when given a constant number of students, we can efficiently solve the most general form of the problem with no restrictions on either the credit caps or the number of credits for each course, and with arbitrary preferences for each student.

Algorithm 1 Round Robin Algorithm for CAUTC-SW.

Require: Set of students $\mathcal{N}$, set of courses $\mathcal{M}$, uniform (unit) utilities
Ensure: Assignment of courses to students.
function RoundRobin($\mathcal{N}$, $\mathcal{M}$)
  Sort $\mathcal{M}$ chronologically by earliest finish time.
  Initialize student assignments $\mathcal{A}$ to empty sets. ▷ each student starts out with no courses
  for course $j \in \mathcal{M}$ in sorted order **do**
    Let $T = \{s \mid |A_s| < C_s, \text{no course in } A_s \text{ conflicts with } j\}$.
    if $|T| > 0$ **then**
      Let $s = \min_{s' \in T} (|A_{s'}|)$ (breaking ties by student index).
      Update $A_s = A_s \cup \{j\}$ ▷ Assign course j to student s
  return $\mathcal{A}$

▶ **Theorem 8.** *CAUTC-SW is polynomial-time solvable when there are only a constant number of students and credit counts for courses can be distinct but are each $O(1)$.*

The proof of Theorem 8 can be found in Appendix B.3.

▶ **Theorem 9.** *Algorithm 1 solves CAUTC-SW in $O((n+m)\log n)$ time when there are (1) uniform credits for all courses, i.e. $c_j = c_{j'}$ for all $j, j' \in \mathcal{M}$, (2) uniform course lengths, i.e., $d_j = d_{j'}$ for all $j, j' \in \mathcal{M}$, and (3) uniform utilities i.e., $u_i(j) = u_{i'}(j)$ for all $i, i' \in \mathcal{N}$.*

We prove Theorem 9 via a variation of the greedy-comes-first strategy; we present our full proof in Appendix B.4. When the durations of the courses are not uniform, we can obtain a (1/2)-approximate allocation for CAUTC-SW.

▶ **Lemma 10.** *There is a $O((n+m)\log n)$ time round-robin algorithm for CAUTC-SW that obtains a $1/2$-approximation when there are (1) n students, (2) uniform credit caps i.e. for any pair of students $i, i' \in \mathcal{N}$, we have $C_i = C_{i'}$, and (3) uniform utilities i.e. for any pair of students $i, i' \in \mathcal{N}$ and jobs $j, j' \in \mathcal{M}$, we have $u_i(j) = u_{i'}(j')$.*

Proof. We use the same algorithm as before, given in Algorithm 1. However, we use a slightly different analysis which is somewhat more complicated than our utility proof before but with the same essential flavor of proof using D_i, J_i, B_i. Namely, the one additional property we prove is that when $|B_i| + |D_i| \geq |J_i|$, our new greedy algorithm will pick $|J_i|$ instead of $B_i \cup D_i$. Suppose for contradiction that i picked $B_i \cup D_i$ instead of J_i, then i must have picked a course with *earlier or the same end time* as each of the courses in J_i. We now show that $|B_i \cup D_i| \geq |J_i|$. We prove this through the classic greedy stays ahead proof technique. If one were to chronologically order $B_i \cup D_i$ by finish time and also chronologically order J_i by finish time, and call the two ordered sets as P and Q, respectively, and let P_i denote the i-th course in set P; we will prove that it is always true that for all indices $i \leq |J|$, $f(P_i) \leq f(Q_i)$, where $f(x)$ means the finish time of course x. Also define the start time function of course x as $s(x)$. The base case of $i = 1$ is obviously true due to the nature of the algorithm. Now for the inductive case, assume inductive hypothesis $f(P_i) \leq f(Q_i)$ and we want to prove $f(P_{i+1}) \leq f(Q_{i+1})$. We know that $f(Q_i) \leq s(Q_{i+1})$. Combining this with the inductive hypothesis, we get $f(P_i) \leq s(Q_{i+1})$, so Q_{i+1} is available for our algorithm to choose, and since our algorithm chooses an available course with the earliest end time, $f(P_{i+1}) \leq f(Q_{i+1})$.

Let's assume for the sake of contradiction that $|J| > |B_i| \cup |D_i|$. Through the same argument as in the inductive case above, say $|B_i| \cup |D_i| = p$, then the start time of Q_{p+1} must have a start time later than the finish time of the last course in $|B_i| \cup |D_i|$, i.e. $s(Q_{p+1}) \geq f(P_p)$, but that means our algorithm would have selected Q_{p+1} (some time) after selecting P_p, a contradiction. ◀

For completeness, we state the following form formulation of CAUTC-SW that is solved via an interval coloring algorithm of Carlisle and Lloyd [14].

▶ **Theorem 11** ([14]). *CAUTC-SW can be solved in polynomial time when there are (1) n students, (2) no credit caps i.e., $C_i = m$, and (3) uniform utilities i.e. for any pair of students $i, i' \in \mathcal{N}$, we have $u_i(j) = u_{i'}(j)$.*

3.3 Guaranteeing Envy-Freeness Up to Any Good

Maximizing seat occupancy is a reasonable objective only from a financial perspective for the university, but oftentimes, maximizing seat occupancy could result in highly unfair schedules for the students. For example, student A might get all of his favorite courses while student B gets none of his desired courses. We, therefore, consider CAUTC-SW under several fairness notions, such as envy-free up to any good (Definition 2) and envy-free up to one good (Definition 3).

▶ **Theorem 12.** *There is an $O((n+m)\log n)$-time algorithm for CAUTC-SW that is EFX when there are (1) n students, (2) uniform credit caps i.e. for any pair of students $i, i' \in \mathcal{N}$, we have $C_i = C_{i'}$, and (3) uniform utilities i.e. for any pair of students $i, i' \in \mathcal{N}$ and any pair of jobs $j, j' \in \mathcal{M}$, we have $u_i(j) = u_{i'}(j')$.*

Proof. Our algorithm is the same round robin algorithm given in Algorithm 1. We first prove the following lemma.

▶ **Theorem 13.** *When student i is no longer able to choose a feasible course, there will be at most $n-1$ courses that can be assigned after i's turn and each of these courses is assigned to a different student.*

Proof. Because utilities are uniform, if student i is no longer able to choose a course, this means that all remaining courses conflict with the courses they are assigned. Suppose the last course that is assigned to student i is course j. Because we are assigning courses in Algorithm 1 in a round robin manner in an order determined by non-decreasing end time, all remaining courses (yet to be considered by the algorithm) that can be assigned have end time no earlier than the end time of j. Let this set of courses be A. Since i is no longer able to receive a course, either there remains only $n-1$ courses or A has at least $n-1$ courses and at least $|A|-n+1$ courses in A all conflict with j. Since all courses in A have end time no earlier than the end time of j, these $|A|-n+1$ courses all conflict with each other. In either of these two cases, at most $n-1$ courses can be assigned after i's turn. Furthermore, these courses are assigned to different students. If there are at most $n-1$ courses in A, then by nature of the algorithm, these courses all have end times later than the end times of courses assigned to students; furthermore, the ending time of the last course assigned to each student can be no later than the end time of j by the nature of our algorithm. Hence, two such courses can be assigned to one student, then one of these courses can be assigned to j. Thus, since we are assigning courses to a student with the fewest number of courses, each of these courses is assigned to a different student. Finally, all additional $|A|-n+1$ courses all conflict with each other and hence no two of these courses can be assigned to the same student. ◀

Hence, by the time the algorithm completes and by Theorem 13, the cardinalities of all students' allocations are within one of each other, therefore achieving EFX. ◀

3.4 Maximizing Max-Min Objective

In this section, we consider the max-min objective, Santa Claus (SC) problem (Definition 5). We first show that our algorithm in Algorithm 1 gives a (1/4)-approximate CAUTC-SC allocation. Specifically, we prove the following.

▶ **Lemma 14.** *There is a $O((n+m)\log n)$ time round robin algorithm (Algorithm 1) for CAUTC-SC that obtains a $(1/4)$-approximation when there are (1) n students, (2) uniform credit caps i.e. for any pair of students $i, i' \in \mathcal{N}$, we have $C_i = C_{i'}$, and (3) uniform utilities i.e. for any pair of students $i, i' \in \mathcal{N}$ and jobs $j, j' \in \mathcal{M}$, we have $u_i(j) = u_{i'}(j')$.*

Proof. Given a set of courses with total utility U, the max-min value of any allocation is at most $\lfloor \frac{U}{n} \rfloor$. We now consider two possible cases with respect to the values of $\lfloor \frac{U}{n} \rfloor$. First, we consider the case when $\lfloor \frac{U}{n} \rfloor \geq 2$. In this case, by Theorem 13, the max-min value of our allocation is at least $\frac{U}{2n} - 1 \geq \frac{U}{4n}$. Now, we consider the case when $\lfloor \frac{U}{n} \rfloor < 2$. In this case, either the max-min value is 0 or the max-min value is 1. If the max-min value is 0, then we trivially obtain our approximation since any allocation will result in the correct approximation. Otherwise, if the max-min value is 1, then there is one student who gets only one course. We show that if the max-min value is 1, then our algorithm also allocates at least one course to every student. The criteria for our algorithm giving one course to each student is that there exists at least n courses. Since our algorithm assigns the courses in a round robin manner, if there are at least n courses, then our algorithm will assign at least one course to each student. In order for the max-min value to be 1, there must exist at least

n courses; hence, the max-min value of allocations given by our algorithm matches that of the value given in OPT. Thus, by the two cases we just showed, the approximation factor is at least $\frac{\frac{U}{4n}}{\frac{U}{n}} = \frac{1}{4}$. ◀

4 Binary Preferences for Classes with Uniform Credits

In this section, we discuss the setting where students have binary preferences for courses. This is a very realistic setting since it is often the case that students want to take certain courses and not others. We denote the binary preferences of the students as $U : \mathcal{N} \times \mathcal{M} \mapsto \{0, 1\}$, where $u_i(c) = 1$ denotes that the student $i \in \mathcal{N}$ wants to take the course c, and $u_i(c) = 0$ denotes that course c is not desired by student i. If a student has $u_i(c) = 1$, then we say that student i *desires* course c; otherwise, we say that student i does not desire course c. Each student i has a credit cap denoted by C_i. In this section, all courses have uniform number of credits; i.e. all courses have the same number of credits. Because of this assumption, we can assume all courses are 1 credit each and we scale the credit caps of each student to the maximum number of courses that can fit in the student's schedule.

4.1 Maximizing Social Welfare

We first present an algorithm that gives an approximation for CAUTC-SW given binary preferences. Our algorithm proceeds as follows. Sort the students by credit cap from largest credit cap to smallest (Line 2). Then, we iterate the following procedure. Let the current student be the first student in the sorted order of the students by credit cap with no assigned courses (Line 4). We find an independent set of maximum size among all courses with non-zero utility for the current student (Line 5). For each independent set I and the associated student $i \in \mathcal{N}$, we sort the courses in I and give the first $\max(|I|, c_i)$ courses in I in the sorted order to student i (Lines 7, 8, 9). Finally, we remove the allocated courses from the set of available courses (Line 10).

Algorithm 2 Binary Utilities Algorithm for CAUTC-SW.

Require: Set of students $\mathcal{N}$, set of courses $\mathcal{M}$, binary utilities U
Ensure: Assignment of courses to students.

```
1:  function MaxIndependentSetRoundRobin(N, M, U)
2:      Sort N in non-increasing order by credit cap.
3:      Initialize student assignments A to empty sets.   ▷ student starts out with no courses
4:      for student i ∈ N in sorted order do
5:          Let I = MIS({j | j ∈ M, u_i(j) > 0}).          ▷ Find MIS in remaining courses.
6:          if |I| > C_i then
7:              Sort I by end time.
8:              Set I ← I[C_i].          ▷ Resize the MIS to be the first C_i courses in the MIS.
9:          Set A_s ← I.
10:         Update M = M \ I.                               ▷ Remove assigned courses.
11:     return A
```

▶ **Theorem 15.** *Algorithm 2 solves CAUTC-SW in $O(n^2)$ time with an $(1/2)$-approximation when there are n students, arbitrary credit caps C_i for all $i \in \mathcal{N}$, unit credits per course $c_j = 1$ for all $j \in \mathcal{M}$, and binary utilities for all students, i.e. $u_i(j) \in \{0, 1\}$ for all $i \in \mathcal{N}$.*

Proof. In the sorted order of courses by end time in I, if course $j \in \mathcal{M}$ is assigned in OPT and by our algorithm, then we skip this course in our analysis. However, if the course is assigned in OPT but not assigned by our algorithm, then we need to argue that either another course is assigned in its place or that we can *charge* it to another assigned course. For all of the below cases, suppose that course $j \in \mathcal{M}$ is assigned to student $i \in \mathcal{N}$ in OPT but not assigned in our assignment. For simplicity, we denote the assignment produced by our algorithm as $\mathcal{A}$. Let D_i be the set of courses assigned to student i in $\mathcal{A}$ which were not assigned to any student in OPT; let B_i be the set of courses assigned to i in $\mathcal{A}$ but assigned to $q \neq i \in \mathcal{N}$ in OPT. Finally, let J_i be the set of courses assigned to i in OPT but assigned to no student in $\mathcal{A}$. We consider all possible cases below.

- If $|D_i| \geq |J_i|$, then for each course in J_i, we can replace it with a course in D_i and achieve the same maximum total utility.
- If $|D_i| < |J_i|$, then we consider two additional cases:
  - It is impossible to have $|B_i| + |D_i| < |J_i|$ since $|J_i|$ is a larger independent set and would have been assigned to i instead of $B_i \cup D_i$.
  - Then, the remaining case is that $|B_i| + |D_i| \geq |J_i|$. This case is the core of our proof. In this case, we know that $|B_i| \geq |J_i| - |D_i|$. We pick an arbitrary set of $|D_i|$ jobs in J_i and replace them each with a unique job in D_i. This does not change the optimum total utility value. Now, we charge each of the remaining $|J_i| - |D_i|$ jobs in J_i to a job in $|B_i|$. We now count the number of "charges" that each course in $|B_i|$ gets. Since $|B_i| \geq |J_i| - |D_i|$ and we do not charge a course in B_i with any other course not in J_i, each course in B_i is charged with at most one charge resulting from a course in J_i.

We now count the number of courses assigned in both OPT and $\mathcal{A}$ as well as the number of charges each course gets. By the cases above, each of these courses gets at most 1 charge. Hence, if each charge is added to the set of allocated courses, the utility increases by at most a factor of two. Hence, our algorithm produces a (1/2)-approximation. ◀

4.2 Guaranteeing Envy-Freeness Up to One Good

Given an allocation of courses to students $\mathcal{A} = (A_1, \ldots, A_i, \ldots, A_n)$ (where A_i is the set of courses assigned to student i), a student i is said to ***envy*** student i' if the number of student i's desirable courses in A_i is less than that in $A_{i'}$, that is, $|\{j \mid u_i(j) > 0, j \in A_i\}| < |\{j \mid u_i(j) > 0, j \in A_{i'}\}|$. Similarly, an allocation A is called EF1 when for every pair of students $i, i' \in \mathcal{N}$, the following holds: $|\{j \mid u_i(j) > 0, j \in A_i\}| \geq |\{j \mid u_i(j) > 0, j \in A_{i'}\}| - 1$. Note that in the binary valuation setting, EF1 implies that, removing *any* course that i desires from $A_{i'}$ results in i no longer envying i'. We provide an algorithm (Algorithm 3) and prove that this algorithm satisfies the stronger fairness criterion called EF1-CC (Definition 4).

Our algorithm is a simple modification of the round-robin algorithm given in Algorithm 1. The only change we make to the algorithm is that when we perform the round-robin assignment, each course is iteratively assigned to only one of those students who have non-zero utility for the course, in addition to ensuring that the selected student has the minimum number of current courses, has not reached credit cap and has no conflict with the course. Our modified pseudocode is given in Algorithm 3.

Specifically, Algorithm 3 first sorts the courses chronologically by finish time (Line 2). Then, we iterate over the courses one by one in the sorted order of finish time (Line 4). Among the students who have non-zero preference for the course, have not reached their credit caps, and have no conflicts with the course (Line 5), we select a student (breaking ties arbitrarily) with the least number of assigned courses among these students (Line 7). Finally, we assign the course to the student (Line 8).

Algorithm 3 Round Robin Algorithm for EF1-CC Allocation with Binary Utilities.

Require: Set of students $\mathcal{N}$, set of courses $\mathcal{M}$, binary utilities U
Ensure: EF1-CC Allocation for Binary Utilities
function EF1CCRoundRobin($\mathcal{N}$, $\mathcal{M}$, U)
  Sort $\mathcal{M}$ chronologically by earliest finish time.
  Initialize student assignments $\mathcal{A}$ to emptysets. ▷ students start out with no courses
  for course $j \in \mathcal{M}$ in sorted order **do**
    Let $T = \{s \mid u_s(j) = 1, |A_s| < C_s, \text{no course in } A_s \text{ conflicts with } j\}$.
    if $|T| > 0$ **then**
      Let $s = \min_{s' \in T} (|A_{s'}|)$ (breaking ties arbitrarily).
      Update $A_s = A_s \cup \{j\}$ ▷ Assign course j to student s
  return $\mathcal{A}$

▶ **Theorem 16.** *Under binary preferences, uniform credits for all courses, and arbitrary credit caps, the round-robin algorithm given in Algorithm 3 produces an EF1 allocation.*

Proof. We prove by induction that for any two students s and s', student s never envies s' by more than one course throughout the entirety of Algorithm 3. The induction is on the finish time of each course in the schedule of s' among the set of courses for which s has non-zero utility, i.e. we induce on the finish times of the set of courses $L = [j \in A_{s'} \mid u_s(j) > 0]$ sorted from earlier to later times. Notice that L is the set of courses assigned to s' that are desired by s, as courses assigned to s' not desired by s cannot make s envy s' and therefore irrelevant to this proof. Now, for each $i \in [|L|]$, we consider the set of courses assigned to both s and s' which has end time no later than the end time of $L[i]$. For simplicity, we use the phrase *by the time course $L[i]$ ends* to mean that we consider the set of courses held by s and s' with end time no later than $L[i]$.

▶ **Lemma 17.** *For each course $L[i]$ for all $i \in [|L|]$, at the time $L[i]$ ends, student s envies s' by at most one course.*

Proof. We prove via induction on the i-th course of L which ends at time e_i. The base case is when $i = 1$. Student s trivially envies s' by at most 1 because if s has no courses by the time course $L[i]$ ends then s will only envy s' by 1; otherwise, s will not envy s'.

We assume for the purposes of induction that s envies s' by at most one course by the time $L[i]$ ends. We now prove that s envies s' by at most one course by the time $L[i+1]$ ends. By our induction hypothesis, there are two cases, when s envies s' by one course when $L[i]$ ends, and when s does not envy s' when $L[i]$ ends. In the latter case, it is only possible for s to envy s' by at most one course by the time $L[i+1]$ ends since s' has gained at most one additional course which s desires by the time $L[i+1]$ ends. Now we prove the former case. Let j be the next course (after the course $L[i]$) that the algorithm considers that is assigned to either student s or s', is desired by s. Then, course j would fit into the current schedule of both s and s', since j starts after the end time of $L[i]$. Suppose for the sake of contradiction that j is assigned to s'. Since we compare the set of courses that end no later than the end time of $L[i]$, if j is assigned to s' then j has start time later than $L[i]$. Student s envies s' by 1 course among the set of courses she received that end no later than $L[i]$. Then, course j is not assigned to s only if s has a conflicting course (since s has fewer courses than s'); however, this contradicts with j being the next course assigned after $L[i]$ to either s or s'. ◀

Now to prove Theorem 16, we use Lemma 17. Specifically, by the time the last course in L ends, student s envies s' by at most one course. Any course in the schedule of s that ends at a time later than this does not increase the envy s feels towards s'. And due to symmetry, s' similarly does not envy s by more than one course. Similarly, any course assigned to s in between the ending times of $L[i]$ and $L[i+1]$ does not increase the envy of s. ◀

▶ **Theorem 18.** *Under binary preferences and uniform credits for all courses, Algorithm 3 produces an EF1-CC allocation.*

Proof. Theorem 16 stated that no student envies another student by more than one course. We are left to show that no student envies charity by more than 1 course. Assume for the sake of contradiction that there is a student s that envies the charity, this means that (1) $|A_s| < C_s$ where c_s is the credit cap for student s, and (2) there is a bigger independent set of courses (name this set I) among the courses assigned to the charity than the number of allocated courses to s, i.e. $|A_s| < |I|$.

First, all courses in I overlap with A_s because if some course $j \in I$ does not conflict with any course in A_s, then our algorithm would have assigned j to s. If we were to sort I and A_s by earliest finish time first and index them by i, observe that for all i, course $A_s[i]$ ends earlier than $I[i]$ due to our algorithm (this can be proven with a very elementary greedy stays ahead induction proof [28]). This means that $|A_s| \geq |I|$ because if there were to be a course $j = E[|A_s| + 1]$, that means j begins after the last course in A_s ends, which means our algorithm would have assigned j to s. ◀

4.3 Maximizing Max-Min Objective

Now, we look at a more general version of CAUTC-SC considering binary utilities and provide the following algorithm that gives a constant factor approximation when the maximum and minimum durations of any course are within a constant factor c of each other. We first describe our algorithm with the pseudocode provided in Algorithm 4. The algorithm proceeds as follows. The courses are sorted by end time (Line 2). Then, in the sorted order of courses, each course is given to a student who has non-zero preference for the course, has not filled up all of their credits (up to their credit cap), has no conflicting courses, and who has the least number of assigned courses among all students who have non-zero preference for the course (Line 6). Suppose we assign course j to a student i. Let d_i be a *dummy course* that we create for each student i. Then, we repeatedly perform the following procedure until no more *augmenting paths* exist (Line 9):

- For each course assigned to student i, draw a directed edge from course j' assigned to student $i' \neq i$ if j conflicts with j' and removing j' means that j does not conflict with any other course assigned to i' and i' has less assigned courses than i (Line 13).
- For each course assigned to student i, draw a directed edge from dummy course $d_{i'}$ to j if j does not conflict with any course assigned to i' and i' has less than or equal to the number of courses assigned to i (Line 14).
- Repeat with the courses assigned to i' and omit all courses assigned to student i from this part of the graph construction.

Once a full directed acyclic graph is drawn using the above procedure, we define an *augmenting path* to be a directed path with the source at a dummy course and sink at a course of i (Line 16). We repeatedly produce a new directed acyclic graph using the above procedure and switch courses between students via an augmenting path until no such augmenting paths remain (Line 18). Then, we proceed with assigning the next item in the sorted order of courses. We prove that our algorithm returns a constant factor approximation of the max-min objective value.

Algorithm 4 Max-Min Assignment of Courses.

Require: Courses $\mathcal{M}$, students $\mathcal{N}$, binary utilities U
Ensure: Approximate max-min allocation J
function FIND-MAX-MIN-ALLOCATION($\mathcal{M}$, $\mathcal{N}$, U)
  Sort courses in $\mathcal{M}$ by end time from earliest to latest.
  $D \leftarrow \emptyset$.
  Let $Q \leftarrow \emptyset$ be a queue of students.
  for each course j in sorted order **do**
    Assign j to student i with minimum number of assigned courses, has not reached credit cap, where $u_i(j) > 0$, and does not have any conflicting courses.
    Add i to the end of Q.
    Set $AugPath \leftarrow True$.
    while $AugPath$ **do**
      while $Q \neq \emptyset$ **do**
        Remove the first student i' from Q.
        for each course j assigned to i' **do**
          Draw directed edge from j' assigned to student b to j if j' conflicts with j, removing j' results in j conflicting with no course assigned to b conflicting with j and b now has less assigned courses than i', and $b \notin D$. Add b to the end of Q.
          Draw a directed edge from d_b to j if student b does not have any courses that conflict with j and b has at most as many courses as i'. Add b to the end of Q.
        $D \leftarrow D \cup i'$.
      Find an augmenting path with source at a dummy course and sink at course assigned to i and reassign courses along augmenting path from sink to source.
      if there is no augmenting path **then**
        $AugPath \leftarrow False$.
  return Allocation of courses to students.

▶ **Theorem 19.** *Algorithm 4 achieves a c-factor approximate solution for CAUTC-SC, where c is the maximum ratio between the durations of any two courses.*

Proof. Let S denote the set of students with the minimum number of assigned courses by our algorithm. We compare the allocations of courses assigned to each of the students in S by our algorithm with the allocation of courses assigned to the students by OPT. Let $i \in S$ be one such student. Let A_i be the set of courses allocated to student i by our algorithm and OPT_i be the set of courses allocated to i by OPT. There are four different types of courses assigned to these students that we are concerned with. Courses assigned to i in A_i and not in OPT_i can only make max-min greater; thus, we do not consider such courses. The same holds for courses assigned in A_i and by OPT to another student. Then, courses assigned by OPT but not assigned to A_i must conflict with at least one other course assigned to i. Hence, such courses can be charged to the course that it conflicts. The conflicting course(s) cannot be assigned in OPT_i; thus, the course in OPT_i can be charged to one of the conflicting courses. The remaining type are courses that are in OPT_i, not in A_i, but are instead assigned to another student by our algorithm. Let j be one such course; then, either

- Course j is assigned to a student i' with *less* assigned courses than i. This scenario is impossible by definition of i as a student with the smallest number of assigned courses.
- Course j is assigned to a student i' with the same or more assigned courses than i. Student i must be assigned a conflicting course to j, as otherwise, when the last course

assigned to i' is assigned to i', course j would have been transferred to i. Suppose first that i' has a greater number of courses than i and i has no conflicting course with j, then this is a contradiction since j would have been eventually transferred to i. Now suppose i has a course that conflicts j. If this conflicting course has an earlier end time than j, then j can be charged to the conflicting course. Furthermore, any course can conflict with at most c different courses assigned to i in OPT by our assumption of the ratio between the longest class and shortest class. Thus, we charge the course to the conflicting course assigned to A_i; at most c such courses can be charged to any course in A_i. ◀

5 Experimental Results

In this section, we present a case study with data derived from MS students at Northwestern. We compare the performance of our algorithms Algorithm 3 and Algorithm 4 to those of optimal integer programs (IP) implemented using Gurobi [21] in Python. There are two integer programs of note: one to get the max-min value, and one to get the assignment maximizing the total social welfare given the max-min value T such that every student must receive at least T courses. We will henceforth refer to both of these integer programs that produce the optima as OPT. We implement Algorithm 3 and Algorithm 4 in Python [1]. In Algorithm 4, after looping through each course, exchange path operations are initiated. The graphs of exchange paths were implemented in NetworkX[22] in Python. The experiments are conducted on a Dell PowerEdge R740 with 2 x Intel Xeon Gold 6140 2.3GHz 18 core 36 threads processors, 192GB RAM, dual 10Gbps and 1Gbps NICs.

The dataset was obtained through a Google Form sent out to Master's students who wished to take computer science courses. They could select and rank up to five courses. Since ordinal preferences are beyond the scope of this paper, we only considered the courses they desire (binary valuations).

Table 1 Comparison of utilities.

Datasets		max-min			total utility		
		OPT	Algorithm 3	Algorithm 4	OPT	Algorithm 3	Algorithm 4
real-world data	dataset	1	1	1	744	624	744
	alteration 1	2	1	2	725	623	725
	alteration 2	3	2	3	686	686	686
	alteration 3	2	2	2	760	760	760
synthetic data	example 1	2	2	2	7	7	6
	example 2	3	1	2	6	3	6
	example 3	4	3	3	8	6	6
	example 4	1	1	1	5	4	5
	example 5	1	1	1	6	5	6
	example 6	4	4	4	12	12	12
	example 7	4	4	4	12	12	12
	example 8	4	4	4	12	12	12

In terms of utility and max-min value, both algorithms incurred similar values as that of OPT. Table 1 compares the max-min value between OPT, Algorithm 3, and Algorithm 4. For almost all instances listed in Table 2, Algorithm 3 was much faster than OPT. Since our input data is not too large, we could compute an optimal assignment by solving the corresponding IP using Gurobi, which is not scalable in general.

Table 2 Comparison of runtimes in milliseconds. All runtimes correspond to instances in the corresponding cells in Table 1. There is only one column under *total utility* because Algorithm 3 and Algorithm 4 are executed only once, as opposed to the two different linear programs of OPT.

Datasets		max-min			total utility
		OPT	Algorithm 3	Algorithm 4	OPT
real-world data	dataset	64.3614	44.893709	789.936638	1.883833
	alteration 1	59.0851	44.675743	751.317634	1.818061
	alteration 2	47.1846	38.609633	490.054614	1.447398
	alteration 3	59.0924	44.469586	1214.057334	1.625204
synthetic data	example 1	3.672	0.405452	1.505546	1.518124
	example 2	0.6444	0.177736	0.831938	0.186128
	example 3	2.3873	0.263971	1.145906	0.552925
	example 4	2.5193	0.200845	1.052790	0.524367
	example 5	3.3109	0.247982	1.344810	0.588912
	example 6	1.1906	0.242676	0.667803	0.232696
	example 7	11609.3298	594.707318	34986.057784	877.306024
	example 8	634809.8065	31554.700315	2278023.952813	9592.727642

The results of our experiments demonstrate the effectiveness of our algorithms. Algorithm 3 was able to give near-optimal solutions with a significantly reduced computational cost compared to integer programming, a traditional method. The reduced runtime is a testament to the effectiveness of the algorithms and their potential for practical implementation. The findings of this study highlight the potential for further improvement and optimization of these algorithms (especially Algorithm 4 since its runtime has much room for improvement) making them an attractive option for real-world applications. Although Algorithm 4 is slower than OPT for some of the tested instances, we believe it will be much faster and more scalable on instances larger than what we tested in our experiments.

6 Conclusions and Future Work

We investigated the problem of allocating conflicting resources across n agents, while taking into account both fairness as well as overall utility of the assignment. While resource allocation is extremely well studied, cases when the resources have conflicts have not been well studied from an algorithmic perspective.

Several generalizations of the course allocation problem open up interesting new directions for the fair allocation literature, such as generalizing utilities beyond additive binary and considering non-uniform credits for different courses. Further, each course may be a corresponding collection of time intervals (instead of a single interval). While we assume that our courses meet once a week, this may not be true for the general case where courses might meet on Tues-Thurs or Mon-Wed or Mon-Wed-Fri. If two courses overlap in any of the time windows then there is an edge in the conflict graph between them. However, such considerations would make the problem more challenging since the corresponding conflict graphs would be more complicated than interval graphs.

Going ahead, there are several directions for future research that can extend and improve upon our approach to course allocation. By addressing these challenges, we can develop more effective and fair algorithms for allocating courses to students, and better meet the diverse and evolving needs of students.

References

1 Fair course allocation implementations. https://github.com/yiduoke/CS-499-Khuller.

2 Miklos Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J Schulman, and Orli Waarts. Fairness in scheduling. *Journal of Algorithms*, 29(2):306–357, 1998.

3 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 31–40, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132522.

4 Nawal Benabbou, Mithun Chakraborty, Ayumi Igarashi, and Yair Zick. Finding fair and efficient allocations for matroid rank valuations. *ACM Transactions on Economics and Computation*, 9(4):1–41, 2021.

5 Mohamed Bendraouche, Mourad Boudhar, and Ammar Oulamara. Scheduling: Agreement graph vs resource constraints. *European Journal of Operational Research*, 240(2):355–360, 2015.

6 Vittorio Bilò, Angelo Fanelli, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli. The price of envy-freeness in machine scheduling. In *Mathematical Foundations of Computer Science 2014: 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II 39*, pages 106–117. Springer, 2014.

7 Arpita Biswas and Siddharth Barman. Fair division under cardinality constraints. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 91–97, 2018.

8 Hans L Bodlaender and Fedor V Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.

9 Hans L Bodlaender, Klaus Jansen, and Gerhard J Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55(3):219–232, 1994.

10 Flavia Bonomo, Sara Mattia, and Gianpaolo Oriolo. Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem. *Theoretical Computer Science*, 412(45):6261–6268, 2011.

11 Peter Brucker and L. Nordmann. The k-track assignment problem. *Computing*, 52(2):97–122, 1994. doi:10.1007/BF02238071.

12 Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.

13 Yinhui Cai, Guangting Chen, Yong Chen, Randy Goebel, Guohui Lin, Longcheng Liu, and An Zhang. Approximation algorithms for two-machine flow-shop scheduling with a conflict graph. In *Computing and Combinatorics: 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings 24*, pages 205–217. Springer, 2018.

14 Martin C. Carlisle and Errol L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3):225–235, 1995. doi:10.1016/0166-218X(95)80003-M.

15 Nina Chiarelli, Matjaž Krnc, Martin Milanič, Ulrich Pferschy, Nevena Pivač, and Joachim Schauer. Fair allocation of indivisible items with conflict graphs. *Algorithmica*, pages 1–31, 2022.

16 Sami Davies, Thomas Rothvoss, and Yihao Zhang. A tale of santa claus, hypergraphs and matroids. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2748–2757. SIAM, 2020.

17 Colin Fisher. *Resource allocation in the public sector: Values, priorities and markets in the management of public services*. Routledge, 2002.

18 Frédéric Gardi. Mutual exclusion scheduling with interval graphs or related classes. part ii. *Discrete applied mathematics*, 156(5):794–812, 2008.

19 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979. URL: http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455.

20 Guilherme C.M. Gomes and Vinicius F. dos Santos. Kernelization results for equitable coloring**this work was partially supported by capes, cnpq, and fapemig. *Procedia Computer Science*, 195:59–67, 2021. Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium. doi:10.1016/j.procs.2021.11.011.

21 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

22 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.

23 Pierre Hansen, Alain Hertz, and Julio Kuplinsky. Bounded vertex colorings of graphs. *Discrete Mathematics*, 111(1-3):305–312, 1993.

24 Halvard Hummel and Magnus Lie Hetland. Fair allocation of conflicting items. *Autonomous Agents and Multi-Agent Systems*, 36(1):8, 2022.

25 Sungjin Im and Benjamin Moseley. Fair scheduling via iterative quasi-uniform sampling. *SIAM Journal on Computing*, 49(3):658–680, 2020.

26 Klaus Jansen. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180(2):71–81, 2003.

27 Henry A Kierstead, Alexandr V Kostochka, Marcelo Mydlarz, and Endre Szemerédi. A fast algorithm for equitable coloring. *Combinatorica*, 30(2):217–224, 2010.

28 Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2006.

29 Daniel Kowalczyk and Roel Leus. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20(4):355–372, 2017.

30 Bo Li, Minming Li, and Ruilong Zhang. Fair allocation with interval scheduling constraints, 2021. doi:10.48550/arXiv.2107.11648.

31 Richard J Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 125–131, 2004.

32 Amin Mallek and Mourad Boudhar. Scheduling on uniform machines with a conflict graph: complexity and resolution. *International Transactions in Operational Research*, 2022.

33 Jakub Marecek and Andrew J Parkes. Semidefinite programming in timetabling and mutual-exclusion scheduling. *arXiv preprint*, 2019. doi:10.48550/arXiv.1904.03539.

34 Vignesh Viswanathan and Yair Zick. Yankee swap: a fast and simple fair allocation mechanism for matroid rank valuations. *arXiv preprint*, 2022. doi:10.48550/arXiv.2206.08495.

A Maximizing b-Matching with Conflicts

In this section, we justify our model of representing courses as interval graphs by showing that the general problem of assigning courses to students is NP-complete when given arbitrary numbers of time segments (or intervals) for each course. Namely, when each course can take place over any arbitrary number of time periods, then the conflict graph can be represented as any general graph. We now discuss the more general problem of assigning resources to agents where in our specific setting, courses can be modeled as resources and students as agents.

One way to view the problem of maximizing the utility of assigning resources to agents, where each agent is assigned a set of non-conflicting resources, is to realize that any agent's allocation is an independent set in the conflict graph. Assigning resources to n agents then becomes a maximum graph coloring problem, where the resources have to be colored with one of n different colors so that no two adjacent resources have the same color, but we simply attempt to maximize the number of colored resources (nodes). If the conflict graph has no restrictions or structure, then even the simplest case becomes NP-hard as we show next.

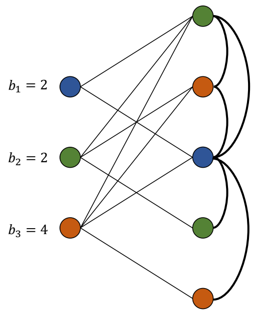

Figure 3 Example b-matching with allocations of resources indicated by the different colors.

A b-matching of any graph is a degree constrained subgraph, where the degree of any node in the subgraph cannot exceed $b(v)$, a specified value. Note that any allocation of goods to agents can be thought of as a b-matching where the edges encode the value of the good to that agent, and the degree constraints model the number of seats in a course (available copies of the good to be assigned to agents) and the degree constraint on the agent nodes corresponds to an upper bound as to how many resources they desire.

▶ **Definition 20** (b-Matching with Conflicts (MbMwC)). *Given a bipartite graph $G = (L \cup R, E)$, a length $|L \cup R|$ vector $\vec{b}$ of non-negative integers, and a set of pairs $(a, a') \in F$ denoting conflicts between nodes on the same side (i.e. either $a \in L$ and $a' \in L$, or $a \in R$ and to $a' \in R$) such that no node v can be matched to a and a' at the same time, a feasible b-matching with conflicts is one where the conflicts are respected and no node p gets matched to more than $b(p)$ nodes on the other side. A maximum b-matching for MbMwC is a feasible matching of maximum weight.*

Even if we simply want to maximize the overall weight of the b-matching (i.e. the sum of everyone's allocation), the problem is NP-hard. This can be shown by a simple reduction from independent set.

▶ **Definition 21** (Maximum Independent Set (MIS)). *Given a graph $G = (V, E)$, set of vertices $V' \subseteq V$ is independent if and only if $\forall p, q \in V', (p, q) \notin E$, i.e. no pair of vertices in V' shares an edge. A maximum independent set of a graph is an independent set with maximum cardinality.*

Given a graph G and an integer k, asking for the existence of an independent set of size at least k is an NP-complete problem. We prove the difficulty of our problem by a reduction from the Independent Set problem.

▶ **Theorem 22.** *Given a bipartite graph $G = (L \cup R, E)$, a vector $\vec{b}$, and a set of pairs F denoting conflicts, finding a b-matching satisfying MbMwC is NP-hard.*

Proof. Given an instance of maximum independent set problem, $G = (V, E)$, and an integer k, we construct an instance of MbMwC, $H = (L \cup R, E')$ where L consists of one node (agent) v and $R = G$. We then create edges from v to all vertices in R. Let $b(v) = k$ and let $b_u = 1$ for all $u \in R$.

If we have a solution to MbMwC in H of weight k, then the matched vertices in R give a maximum independent set in G of cardinality k. In addition, if the graph G does contain an independent set of size at least k then any subset of k nodes can be safely matched with v (and they form a conflict free set). ◀

B Proofs

B.1 Proof of Theorem 6

Proof. This proof is a reduction from 3-PARTITION [19].

▶ **Definition 23** (CAUTC-DECISION). *Consider our problem CAUTC-SW in Section 2.1, instead of the objective of maximizing it, the decision version of it is that given the extra parameter k, is there an allocation such that total student utility is k?*

▶ **Definition 24** (3-PARTITION). *Given a multiset of numbers, can one partition the numbers into triplets such that the sum of each triplet is equal? More precisely, and with an additional restriction on each number. Given a multiset S of $3m$ positive integers where $\sum_{i \in S} x_i = mT$, and each integer $x_i \in S$ satisfies $T/4 < x_i < T/2$, does there exist a partition of S into m disjoint subsets S_1, S_2,...,S_m such that the sum of the x_i values in each set S_j add exactly to T?*

Given an instance of 3-PARTITION, one can reduce it to an instance of CAUTC-DECISION where utilities are uniform and credit caps are uniform and course credit counts are arbitrary. Let there be m students s_1, s_2,...,s_m, each with credit cap T, and let each number $x_i \in S$ from 3-PARTITION represent a course of credit count x_i. No two courses overlap. Every student is interested in every course with uniform utilities. Let $k = mT$. If the solution to CAUTC-DECISION is yes, then the solution to 3-PARTITION is also yes. But first, we have to prove that if there is a solution to CAUTC-DECISION, then each student is allocated exactly three courses. Since the total student utility is $k = mT$ and each student has a credit cap of T, each student is allocated courses whose credits sum to exactly T. Each student must have at least three courses, because each course j has credit $c_j < T/2$. On the other hand, each student must have at most three courses, because each course j has credit $c_j > T/4$. CAUTC-DECISION is therefore NP-hard.

◀

B.2 Proof of Theorem 7

Proof. This proof is based on the reduction from ARC COLORING to the k-track assignment problem by Brucker and Nordmann [11] showing NP-hardness of the k-track assignment problem.

▶ **Definition 25** (k-coloring problem for circular arc graphs (ARC COLORING)). *Given a positive integer k and a set F of n circular arcs $A_1, A_2, ...A_n$, where each A_i is an ordered pair (a_i, b_i) of positive integers where either $a_i < b_i$ or $b_i < a_i$, can F be partitioned into k disjoint subsets so that no two arcs in the same subset intersect?*

The following simple reduction from ARC COLORING shows that CAUTC is NP-hard: we cut the circle from the k-coloring problem for circular arc graphs at some arbitrary but fixed point t. Without loss of generality we calibrate that as $t = 0$, and the courses I_i have the form $I_i = [s_i, t_i]$, where each s_i and t_i is modulo L, the length of the circle.

Now assume that only the courses $I_1, ..., I_r$ contain the point $t = 0$ and that $r \leq k$, for if $r > k$, then the k-coloring problem has no solution. We define k students by making them have a utility of 1 only for the courses that overlap with the time interval $[t_j, s_j]$ for $j = 1, ..., r$ and $[0, L]$ for $j = r + 1, ...k$. Now the problem of assigning the remaining courses $I_{r+1}, ..., I_n$ to these k students is equivalent to the k-coloring problem. ◀

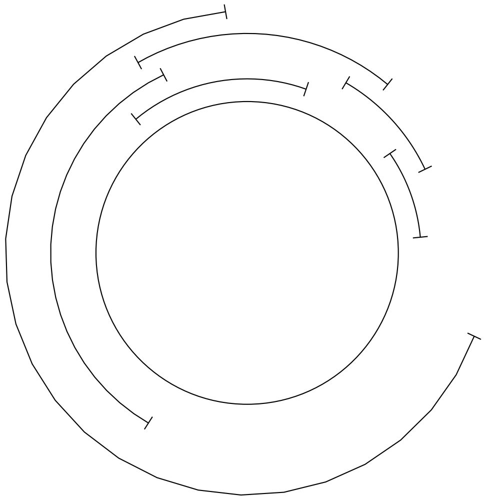

Figure 4 A circular arc model.

B.3 Proof of Theorem 8

Proof. We give a dynamic programming solution for two students, which is easily extendable to any constant k number of students. We sort the courses by non-decreasing start time and use this order to consider the courses in our DP. We define $N(j)$ to be the set of courses that overlap with course j. Given an instance of CAUTC with a constant number of students, for each course $j \in [m]$, course j is either assigned to student 1, to student 2, or to no one. The states of our DP are as follows. For each of the two students, we maintain a counter, p_1 and p_2, respectively, for the remaining number of credits available to student 1 and 2; we also maintain the set of courses available to students 1 and 2 where t_1 and t_2 denote the earliest time that a course which starts at that time can be assigned to students 1 and 2, respectively. Finally, we maintain a counter j indicating the current course being iterated on.

Each time a course j is assigned to wlog student 1, we subtract the credit count of the course, c_j, from p_1 (the total credit count of the student course j is assigned to), increment t_1 by the duration of course j, that is we update t_1 to $t_1 + d_j$. We define our base case to be

$$OPT[p_1, p_2, t_1, t_2, m+1] = 0 \tag{1}$$

for any valid p_1, p_2, t_1, t_2 and our initial state is

$$OPT[p_1, p_2, 0, 0, 0]. \tag{2}$$

We therefore have our recurrence scheme as follows:

$$\begin{aligned} OPT[p_1, p_2, t_1, t_2, j] = \max(&OPT[p_1, p_2, t_1, t_2, j+1], \\ &\mathbb{1}(start_j \geq t_1 \cap c_j \leq p_1) \times (u_1(j) + OPT[p_1 - c_j, p_2, end_j, t_2, j+1]), \\ &\mathbb{1}(start_j \geq t_2 \cap c_j \leq p_2) \times (u_2(j) + OPT[p_1, p_2 - c_j, t_1, end_j, j+1])) \end{aligned} \tag{3}$$

We now prove the optimality of our solution via induction. In the base case, course $m+1$ does not exist, hence, no utility is given for the base case. We now assume for our induction hypothesis that the state for the j-th job is an optimum assignment of courses to students for all valid values of p_1, p_2, t_1, t_2. Now, we show that the optimum solution is computed for the $(j+1)$-st job. For the $(j+1)$-st course, it can either be given to student 1 or 2 or given to no one. Wlog suppose the $(j+1)$-st course is given to student 1. In this case, if

$start_{j+1} < t_1$ or $c_j > p_1$, then the returned value is 0 since course $j+1$ cannot be assigned to student 1 in this case. Otherwise, we show that the states are correctly updated. When $j+1$ is assigned to student 1, the amount of available credits is decreased for student 1 by c_{j+1} and t_1 is increased to end_{j+1}. Since the courses are sorted in non-decreasing order by start time, when course $j+1$ is being considered, no course with start time earlier than $start_{j+1}$ is being considered. Thus, all courses $j' > j+1$ have start time $\geq start_{j+1}$ and so will conflict with course $j+1$ if and only if $start_{j+1} \leq start_{j'} < end_{j+1}$. Hence, setting t_1 to end_{j+1} precisely eliminates the courses $j' > j+1$ that conflict with course $j+1$. Since course $j+1$ has been assigned to student 1, the utility $u_1(j+1)$ is added. Finally, the counter is incremented to $j+2$. The case for assigning $j+1$ to student 2 is symmetric. When $j+1$ is not given to either student, then no utility is added to the previous values and the counter is incremented to $j+2$ with no other changes in the state. There are only three different cases for course $j+1$: it is assigned to either student 1 or 2 or assigned to no one. Using the induction hypothesis and taking the maximum of the three options results in the maximum value for assigning course $j+1$.

Now we prove the runtime of our DP algorithm. Since $c_j = O(1)$ for all $j \in [m]$, we can upper bound p_1 and p_2 by $O(m)$. We can bound t_1 and t_2 as follows. We only increment each of these counters to an end time of a course. There are at most m distinct end times and thus the total number of values t_1 and t_2 can take is m. Finally, the last counter is upper bounded by m. Hence, there are at most $O(m^5)$ different unique states for our DP and our algorithm takes $O(m^5)$ time. For $s = O(1)$ students, our algorithm would take $O(m^{2s+1})$ time. ◀

B.4 Proof of Theorem 9

Proof. We first prove the optimality of Algorithm 1. In this proof, we use the classical greedy-comes-first strategy. In the sorted order of courses by end time, let J be an optimum assignment of courses to students. We show that our greedy algorithm does not produce a worse assignment than J, thus proving its optimality. We prove this via induction on the k-th course in the order sorted by end time. We aim to show that for all $k \leq m$, the number of courses assigned by the greedy algorithm to each student up to course k is at least the number of courses with index $\leq k$ (in the sorted order) assigned in J to each student.

In the base case, when $k = 1$, no courses have been assigned yet, so either the first course is assigned to some student with a sufficiently large credit cap or no student has a sufficiently large credit cap in which case it also cannot be assigned in J. We assume for our induction hypothesis that our greedy algorithm has assigned at least as many courses up to and including the k-th course to each student as the number of courses in J with index $\leq k$ (in the sorted order by end time) assigned to each student. We now prove this for the $(k+1)$-st course. The trivial cases are when the $(k+1)$-st course is not in J or if the $(k+1)$-st course is assigned by the greedy algorithm. Let the $(k+1)$-st course be course j. If the course is in J and it is not assigned by the greedy algorithm to any student, then each student must satisfy at least one of the two following scenarios:

1. Student $i \in [n]$ has not enough remaining credits.
2. Student $i \in [n]$ is assigned a conflicting course.

If Item 1 is true, then student i is assigned as many courses by the greedy algorithm as they were assigned in J; in other words, student i is assigned the maximum number of courses they can take; this means that the greedy algorithm returned a solution no worse than J, since every student has reached their credit cap (since all courses have the same number of credits), and there is no way to improve upon that.

Otherwise, if Item 1 is not true and Item 2 is true then we consider the course with the *latest* end time that is $\geq end_j$. Such a course must exist by our greedy algorithm since if no such conflicting course exists, then j would be assigned to i. Let this conflicting course be j'. Then, courses j and j' cannot both be assigned to student i in J. By our induction hypothesis, the greedy algorithm assigned at least as many courses to student i with index $\leq k$ as the number of courses assigned to i in J with index $\leq k$. Suppose wlog that j' is the only course assigned to i that conflicts with j and we remove course j' from student i's assignment and instead assign j. Then, the number of courses assigned to i cannot increase. Now we argue that removing j' cannot allow another course to be assigned to i. Suppose there exists another course ℓ that is assigned in J and conflicts with j' and does not conflict with j (so that both ℓ and j can be assigned to i if j' is removed). Since all courses have the same duration, it must be the case that if ℓ exists then ℓ has start time earlier than j' and has end time earlier than the start time of j. In that case, j' could not have prevented ℓ from being assigned to i and there exists another course assigned by greedy to i that conflicts with ℓ. Hence, no such ℓ can exist and removing j' and adding j cannot lead to another course ℓ with start time earlier than $start_j$ to be assigned to i. In other words, if ℓ had been assigned to i by greedy, then j would also have been chosen, which contradicts our initial assumption that j and ℓ conflict; and if ℓ hadn't been assigned to i by greedy, it's because a course that starts earlier than ℓ overlaps with it, in which case removing j' does not enable ℓ to be assigned to i.

Finally, courses j' and j cannot both be assigned to the same student in J. Thus, if j' is assigned to a student in J, then j is not assigned to that student. Hence, we only need to consider the case when j' is not assigned in J. By our argument above, at most one course in J is charged to a course assigned by our algorithm; hence, in this case, by what we proved above and by the induction hypothesis the number of courses assigned to i by the greedy algorithm with index $\leq k+1$ is at least the number of courses assigned to i with index $\leq k+1$ by the optimum solution J. ◀