

A Heuristic Approach to Optimize Multidimensional Scheduling

Liam Resnick - 11th Grade
Lower Moreland High School

Definitions

Heuristics - Approach to problem-solving in which the objective is to produce a working solution within a reasonable time frame.

CNF formula - Approach to Boolean logic that expresses formulas as conjunctions of clauses with an AND or OR.

Literals - Synthetic representations of boolean, character, numeric, or string data.

DIMACS CNF - A formatting option for textually representing a formula in conjunctive normal form.

SAT Solvers - Take CNF formulas as inputs and output either a satisfying Boolean assignment or UNSAT if it is unable.

Research

- Every year, schools must assign courses to students adhering to complicated criteria which, when done manually, requires an enormous amount of time, and even when assisted by technology, limits schools' resources.
- Modern SAT solvers are extremely efficient and can often solve problems involving millions of clauses in practice.
- In order to use a SAT solver, one must first convert criteria into boolean logic and CNF so that the conditions are more amenable to algorithmic manipulation.

$$\alpha = P \vee (Q \rightarrow R)$$

$$\neg \alpha = \neg (P \vee (Q \rightarrow R))$$

$$\equiv \neg (P \vee (\neg Q \vee R)) \quad \text{Implication law}$$

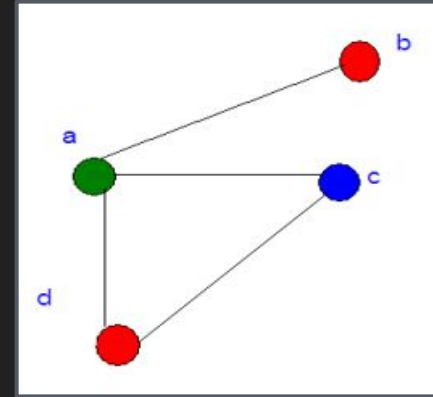
$$\equiv \neg P \wedge (\neg(\neg Q \vee R)) \quad \text{De-Morgans law}$$

$$\equiv \neg P \wedge (Q \wedge \neg R) \quad \text{De-Morgans law \& Double negation}$$

$$\therefore \alpha \equiv \neg P \wedge (Q \wedge \neg R)$$

Understanding SAT

- Problem: Assign a colour to each vertex of a graph with the restriction that two adjacent vertices are not the same colour.
 1. Input number of nodes, edges, colours
 2. Identify of variables and construct the clauses
 3. Assign truth values to literals
 4. Check that each clause is valid
 5. Repeat Steps 3 and 4 until clauses are true
- Type 1 clauses:
 - $(\sim a1 \vee \sim c1) (\sim a2 \vee \sim c2) (\sim a3 \vee \sim c3)$
- Type 2 clause:
 - $(a1 \vee a2 \vee a3)$
- Type 3 clauses:
 - $(\sim a1 \vee \sim a2 \sim a1 \vee \sim a3 \sim a3 \vee \sim a2)$



SI No.	Clauses	SI No.	Clauses	SI No.	Clauses	SI No.	Clauses
1	$\sim a1 \vee \sim c1$	8	$\sim d2 \vee \sim c2$	15	$c1 \vee c2 \vee c3$	22	$\sim b2 \vee \sim b3$
2	$\sim a2 \vee \sim c2$	9	$\sim d1 \vee \sim c1$	16	$d1 \vee d2 \vee d3$	23	$\sim c1 \vee \sim c2$
3	$\sim a3 \vee \sim c3$	10	$\sim a1 \vee \sim b1$	17	$\sim a1 \vee \sim a2$	24	$\sim c1 \vee \sim c3$
4	$\sim a1 \vee \sim d1$	11	$\sim a2 \vee \sim b2$	18	$\sim a1 \vee \sim a3$	25	$\sim c2 \vee \sim c3$
5	$\sim a2 \vee \sim d2$	12	$\sim a3 \vee \sim b3$	19	$\sim a3 \vee \sim a2$	26	$\sim d1 \vee \sim d2$
6	$\sim a3 \vee \sim d3$	13	$a1 \vee a2 \vee a3$	20	$\sim b1 \vee \sim b2$	27	$\sim d1 \vee \sim d3$
7	$\sim d3 \vee \sim c3$	14	$b1 \vee b2 \vee b3$	21	$\sim b1 \vee \sim b3$	28	$\sim d2 \vee \sim d3$

Background & Expected Outcome

- The work I did can be broken down into 3 parts:
 - Figuring out the implicit constraints of scheduling and producing a system for expressing students and courses through boolean logic
 - Writing a program which auto-generates clauses in DIMACS CNF format
 - Using SAT4j to generate a schedule and interpreting the numerical output into courses within a schedule
- Ideally, the model functions as hypothesized and schedules students according to implicit constraints. Further, in my attempt to optimize the scheduling process, I hope to find an approach with a heightened runtime which would sort faster and therefore be more cost-efficient and viable.

As an example the formula $(x \vee y \vee \neg z) \wedge (\neg y \vee z)$ could be encoded as this:

```
p cnf 3 2
1 2 -3 0
-2 3 0
```

Hypothesis & Null

- Hypothesis: With a proper encoding, I expect that SAT4j (a satisfiability solver) will produce schedules in a reasonable time at different scales / school sizes based on layered out criteria for the schedules.
- Null: Once SAT4j creates the schedules, there will be no disparity or a negative disparity between the time that a SAT solver creates the schedules and the time it would take to brute-force schedules.



Variables

Independent:

- The method used for creating various schedules for students based on criteria (SAT solver vs brute-force algorithms / manual)

Dependent:

- The time it takes to create schedules and the accuracy of those schedules

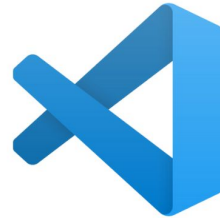
Control:

- The schedules are based on auto-generated, basic clauses that could represent specific classes within a real school system
- The level of computation provided to the SAT solver (to not unintentionally affect performance)

Material list

Materials needed for the experiment:

- Laptop running windows with a core I7 9th generation processor
- Java 8
- Text Editor - VS Code
- Powershell
- Imports: matplotlib, files, sklearn



Procedure

- Research state of the art satisfiability solvers
- Research / create some of the criteria for scheduling in CNF
- Generate simulated students and courses and explore ways to express both numerically (DIMACS)
- Create mathematical models to encode criteria through conditionals
 - Students can't take multiple classes at the same time
 - Students must take a class every period
 - Students can only take a class once a day (even if it's offered more)
- Use SAT-solvers to optimize schedule making
- Create a reader to interpret the results and output a schedule
- Compare using a SAT solver to an alternative method to quantify any improvement

Code (error)

```
for(int i = 0; i < courseNum; i++) {
    int temp = -1;
    while(temp < 0) {
        temp = 1;
        Random r = new Random();
        char Course = (char)(r.nextInt(bound: 10) + 'a');
        int Period = (int)(Math.random()*(8)) + 1;
        String y = Course + "" + Period;

        for (String x : names) {
            if(x.equals(y)) {
                temp = -1;
            }
        }
        courseName = y;
    }

    names.add(courseName);
    //Not sure about the naming of the next 2 lines
    Course y = new Course(courseName);
    instances.add(y);
}

multiClass(instances);
```

```
static void multiClass(ArrayList<Course> courseList) {
    for(int i = 1; i <= 8; i++) {
        int counter = 0;
        int z = 0;
        for (Course x : courseList) {
            if(x.getName().indexOf(str: "1") != -1) {
                counter++;
            }
        }

        int[] arr = new int[counter];
        for (Course x : courseList) {
            if(x.getName().indexOf(str: "1") != -1) {
                arr[z] = makeNum(x.getName());
                z++;
            }
        }

        sort(arr);

        for(int j = 0; j < arr.Length; j++) {
            System.out.print(arr[j] + " ");
        }
    }
}
```

```
static int makeNum(String x) {
    char char1 = x.charAt(index: 0);
    int y = char1 - 96;
    Integer j = new Integer(y);
    int number = Integer.parseInt
        (j.toString() + x.charAt(index: 1));
    return number;
}

static int[] sort(int[] a) {
    int n = a.Length;
    for (int i = 0; i < n-1; i++)
    {
        int min_idx = i;
        for (int j = i+1; j < n; j++) {
            if (a[j] < a[min_idx]) {
                min_idx = j;
            }
        }
        int temp = a[min_idx];
        a[min_idx] = a[i];
        a[i] = temp;
    }
    return a;
}
```

Code - Main

```
//this loop deals with people ([1-80], [81-160], [161-240]...)
for(int i = 0; i < instances.size(); i++) {
    //These loops deal with the individual person
    for(int z = 0; z < 8; z++) {
        for(int j = instances.get(i).getArr()[z]; j <= instances.get(i).getArr()[79]; j += 8) {
            for(int k = j + 8; k <= instances.get(i).getArr()[79]; k += 8) {
                complete += ((-1 * j) + " " + (-1 * k) + " " + 0);
                complete += ("\n");
            }
        }
    }
}

for(int i = 0; i < instances.size(); i++) { //dealing with first person
    for(int z = 0; z < 8; z++) { // dealing with individual period
        for(int j = instances.get(i).getArr()[z]; j <= instances.get(i).getArr()[79]; j += 8) {
            complete += ((j) + " ");
        }
        complete += ("0\n");
    }
}

for(int i = 0; i < instances.size(); i++) {
    for(int z = 0; z <= 79; z += 8) {
        for(int j = instances.get(i).getArr()[z]; j <= instances.get(i).getArr()[z + 7]; j += 1) {
            for(int k = j + 1; k <= instances.get(i).getArr()[z + 7]; k += 1) {
                complete += ((-1 * j) + " " + (-1 * k) + " " + 0);
                complete += ("\n");
            }
        }
    }
}
```

cnf.txt

1	p	cnf	400	3240
2	-1	-9	0	
3	-1	-17	0	
4	-1	-25	0	
5	-1	-33	0	
6	-1	-41	0	
7	-1	-49	0	
8	-1	-57	0	
9	-1	-65	0	
10	-1	-73	0	
11	-9	-17	0	
12	-9	-25	0	
13	-9	-33	0	
14	-9	-41	0	
15	-9	-49	0	
16	-9	-57	0	
17	-9	-65	0	
18	-9	-73	0	
19	-17	-25	0	
20	-17	-33	0	
21	-17	-41	0	
22	-17	-49	0	
23	-17	-57	0	
24	-17	-65	0	
25	-17	-73	0	
26	-25	-33	0	
27	-25	-41	0	
28	-25	-49	0	
29	-25	-57	0	

Code - Reader

Run | Debug

```
public static void main(String str[]) throws IOException {
    Scanner scan = new Scanner(System.in);
    System.out.println(x: "SAT4j output? ");
    String output = scan.nextLine();
    System.out.println(x: "How many students are there? ");
    int z = scan.nextInt();

    String[] arr = new String[z * 8]; // each person has total 8 classes
    String temp = "";
    int counter = -1;

    for(int i = 0; i < output.length(); i++) {
        if(i + 1 == output.length() || i + 2 == output.length()) {
            break;
        }
        if(output.substring(i, i + 1).equals(anObject: " ") &&
            !(output.substring(i + 1, i + 2).equals(anObject: "-"))) {
            int x = i + 1;
            if(x >= output.length()) {
                break;
            }
            counter++;
            while(!(output.substring(x, x + 1).equals(anObject: " "))) {
                temp += output.substring(x, x + 1);
                x++;
            }
            arr[counter] = temp;
            temp = "";
        }
    }
}
```

```
String courseName = "";
int p = 80;
for(int i = 1; i <= z; i++) {
    System.out.print("\nStudent " + i + " schedule is: ");

    for(int j = (i - 1) * 8; j < (i * 8); j++) {
        courseName += (Integer.parseInt(arr[j]) / p) + 1;
        int r = Integer.parseInt(arr[j]) -
            (80 * (Integer.parseInt(arr[j]) / p));
        if(r <= 8) {
            courseName += "A";
        } else if(r <= 16) {
            courseName += "B";
        } else if(r <= 24) {
            courseName += "C";
        } else if(r <= 32) {
            courseName += "D";
        } else if(r <= 40) {
            courseName += "E";
        } else if(r <= 48) {
            courseName += "F";
        } else if(r <= 56) {
            courseName += "G";
        } else if(r <= 64) {
            courseName += "H";
        } else if(r <= 72) {
            courseName += "I";
        } else {
            courseName += "J";
        }
    }
}
```

Code - "Brute force"

```
public static void main(String str[]) throws IOException {
    final long startTime = System.nanoTime();
    Scanner scan = new Scanner(System.in);
    System.out.println(x: "How many students are there? ");
    int z = scan.nextInt();
    String[] arr = new String[z * 80];
    for(int i = 0; i < arr.length; i++) {
        arr[i] = "";
```

```
        ArrayList<String> finallist = new ArrayList<String>();
        for(int i = 1; i < arr.length; i++) {
            if(Integer.parseInt(arr[i].substring(arr[i].length()-1)) == i % 8) {
                finallist.add(arr[i]);
                i += 8;
            }
        }

        for (String p : finallist) {
            System.out.print(p + " ");
        }

        final long duration = System.nanoTime() - startTime;
        System.out.println(duration);
```

```
    for(int i = 0; i < arr.length; i++) {
        arr[i] += (i / 80) + 1;

        int r = (i % 80);
        if(r <= 8) {
            arr[i] += "a";
        } else if(r <= 16) {
            arr[i] += "b";
        } else if(r <= 24) {
            arr[i] += "c";
        } else if(r <= 32) {
            arr[i] += "d";
        } else if(r <= 40) {
            arr[i] += "e";
        } else if(r <= 48) {
            arr[i] += "f";
        } else if(r <= 56) {
            arr[i] += "g";
        } else if(r <= 64) {
            arr[i] += "h";
        } else if(r <= 72) {
            arr[i] += "i";
        } else {
            arr[i] += "j";
        }
    }
```

```
        if(r % 8 == 0) {
            arr[i] += 8;
        } else if(r % 8 == 1) {
            arr[i] += 1;
        } else if(r % 8 == 2) {
            arr[i] += 2;
        } else if(r % 8 == 3) {
            arr[i] += 3;
        } else if(r % 8 == 4) {
            arr[i] += 4;
        } else if(r % 8 == 5) {
            arr[i] += 5;
        } else if(r % 8 == 6) {
            arr[i] += 6;
        } else if(r % 8 == 7) {
            arr[i] += 7;
        }
    }
```


Reader Results

```
-278 -279 -280 -281 -282 -283 -284 -285 -286 -287 -288 -2
320 -321 322 -323 -324 -325 -326 -327 -328 329 -330 -331
-363 -364 365 -366 -367 -368 -369 -370 -371 -372 -373 -3
How many students are there?
5
```

```
Student 1 schedule is: 1A2 1C8 1E6 1F3 1G5 1H7 1I4 1J1
Student 2 schedule is: 2B7 2D2 2E8 2F3 2G5 2H6 2I4 2J1
Student 3 schedule is: 3A5 3B3 3C6 3D1 3E2 3H8 3I7 3J4
Student 4 schedule is: 4A4 4B3 4C6 4D8 4E1 4G7 4H5 4J2
Student 5 schedule is: 5A2 5B1 5C6 5D4 5E3 5F5 5G7 5I8
PS C:\Users\liamr\Desktop\PJAS 2023> █
```

```
Student 1 schedule is: 1A3 1B6 1D2 1E8 1G5 1H7 1I1 1J4
Student 2 schedule is: 2B6 2C1 2D5 2E7 2F3 2G4 2I2 3A8
Student 3 schedule is: 3B2 3C6 3E1 3F8 3G7 3H3 3I5 3J4
Student 4 schedule is: 4B1 4C4 4E3 4F2 4G7 4H5 4I8 4J6
Student 5 schedule is: 5A2 5B1 5C5 5D7 5E6 5F3 5H4 5I8
Student 6 schedule is: 6A5 6B1 6C6 6D2 6E3 6G8 6H7 6I4
Student 7 schedule is: 7A1 7B4 7C5 7D6 7E2 7H3 7I7 8A8
Student 8 schedule is: 8A8 8B2 8C6 8D1 8E3 8F5 8H7 8J4
Student 9 schedule is: 9A5 9B1 9C6 9D3 9E2 9G8 9H7 9I4
Student 10 schedule is: 10A2 10B1 10C6 10D4 10E3 10G7 10I8 10J5
```

SAT4j vs “Brute Force” Results

```
c org.sat4j.minisat.constraints.cnf.OriginalBinaryClause => 64000
c org.sat4j.minisat.constraints.cnf.OriginalWLCClause => 800
c 64800 constraints processed.
s SATISFIABLE
```

```
-7722 -7723 -7724 -7725 -7726 -7727 -7728 -7729 -7730
-7753 -7754 -7755 -7756 -7757 -7758 -7759 -7760 -7761
-7784 7785 -7786 -7787 -7788 -7789 -7790 -7791 -7792
-7815 -7816 -7817 -7818 -7819 7820 -7821 -7822 -7823
-7845 -7846 -7847 -7848 -7849 -7850 -7851 7852 -7853
-7876 -7877 -7878 -7879 -7880 -7881 -7882 -7883 -7884
-7906 -7907 -7908 -7909 -7910 7911 -7912 -7913 -7914
-7937 7938 -7939 -7940 -7941 -7942 -7943 -7944 -7945
-7968 -7969 -7970 -7971 -7972 -7973 -7974 -7975 7976
-7998 7999 -8000 0
c Total wall clock time (in seconds) : 0.583
```

```
7787 -7788 -7789 -7790 -7791
-7818 -7819 7820 -7821 -7822
-7848 -7849 -7850 -7851 7852
-7879 -7880 -7881 -7882 -7883
-7909 -7910 7911 -7912 -7913
7940 -7941 -7942 -7943 -7944
-7971 -7972 -7973 -7974 -7975

time (in seconds) : 0.54
```

```

0004 0005 0006 0007 0008 0009 0010 0011 0012 0013 0014 0015 0016 0017
0018 0019 0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 0030 0031
0032 0033 0034 0035 0036 0037 0038 0039 0040 0041 0042 0043 0044 0045
0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059
0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 0070 0071 0072 0073
0074 0075 0076 0077 0078 0079 0080 0081 0082 0083 0084 0085 0086 0087
0088 0089 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099 0100 0101
0102 0103 0104 0105 0106 0107 0108 0109 0110 0111 0112 0113 0114 0115
0116 0117 0118 0119 0120 0121 0122 0123 0124 0125 0126 0127 0128 0129
0130 0131 0132 0133 0134 0135 0136 0137 0138 0139 0140 0141 0142 0143
0144 0145 0146 0147 0148 0149 0150 0151 0152 0153 0154 0155 0156 0157
0158 0159 0160 0161 0162 0163 0164 0165 0166 0167 0168 0169 0170 0171
0172 0173 0174 0175 0176 0177 0178 0179 0180 0181 0182 0183 0184 0185
0186 0187 0188 0189 0190 0191 0192 0193 0194 0195 0196 0197 0198 0199
0200 0201 0202 0203 0204 0205 0206 0207 0208 0209 0210 0211 0212 0213
0214 0215 0216 0217 0218 0219 0220 0221 0222 0223 0224 0225 0226 0227
0228 0229 0230 0231 0232 0233 0234 0235 0236 0237 0238 0239 0240 0241
0242 0243 0244 0245 0246 0247 0248 0249 0250 0251 0252 0253 0254 0255
0256 0257 0258 0259 0260 0261 0262 0263 0264 0265 0266 0267 0268 0269
0270 0271 0272 0273 0274 0275 0276 0277 0278 0279 0280 0281 0282 0283
0284 0285 0286 0287 0288 0289 0290 0291 0292 0293 0294 0295 0296 0297
0298 0299 0300 0301 0302 0303 0304 0305 0306 0307 0308 0309 0310 0311
0312 0313 0314 0315 0316 0317 0318 0319 0320 0321 0322 0323 0324 0325
0326 0327 0328 0329 0330 0331 0332 0333 0334 0335 0336 0337 0338 0339
0340 0341 0342 0343 0344 0345 0346 0347 0348 0349 0350 0351 0352 0353
0354 0355 0356 0357 0358 0359 0360 0361 0362 0363 0364 0365 0366 0367
0368 0369 0370 0371 0372 0373 0374 0375 0376 0377 0378 0379 0380 0381
0382 0383 0384 0385 0386 0387 0388 0389 0390 0391 0392 0393 0394 0395
0396 0397 0398 0399 0400 0401 0402 0403 0404 0405 0406 0407 0408 0409
0410 0411 0412 0413 0414 0415 0416 0417 0418 0419 0420 0421 0422 0423
0424 0425 0426 0427 0428 0429 0430 0431 0432 0433 0434 0435 0436 0437
0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0449 0450 0451
0452 0453 0454 0455 0456 0457 0458 0459 0460 0461 0462 0463 0464 0465
0466 0467 0468 0469 0470 0471 0472 0473 0474 0475 0476 0477 0478 0479
0480 0481 0482 0483 0484 0485 0486 0487 0488 0489 0490 0491 0492 0493
0494 0495 0496 0497 0498 0499 0500 0501 0502 0503 0504 0505 0506 0507
0508 0509 0510 0511 0512 0513 0514 0515 0516 0517 0518 0519 0520 0521
0522 0523 0524 0525 0526 0527 0528 0529 0530 0531 0532 0533 0534 0535
0536 0537 0538 0539 0540 0541 0542 0543 0544 0545 0546 0547 0548 0549
0550 0551 0552 0553 0554 0555 0556 0557 0558 0559 0560 0561 0562 0563
0564 0565 0566 0567 0568 0569 0570 0571 0572 0573 0574 0575 0576 0577
0578 0579 0580 0581 0582 0583 0584 0585 0586 0587 0588 0589 0590 0591
0592 0593 0594 0595 0596 0597 0598 0599 0600 0601 0602 0603 0604 0605
0606 0607 0608 0609 0610 0611 0612 0613 0614 0615 0616 0617 0618 0619
0620 0621 0622 0623 0624 0625 0626 0627 0628 0629 0630 0631 0632 0633
0634 0635 0636 0637 0638 0639 0640 0641 0642 0643 0644 0645 0646 0647
0648 0649 0650 0651 0652 0653 0654 0655 0656 0657 0658 0659 0660 0661
0662 0663 0664 0665 0666 0667 0668 0669 0670 0671 0672 0673 0674 0675
0676 0677 0678 0679 0680 0681 0682 0683 0684 0685 0686 0687 0688 0689
0690 0691 0692 0693 0694 0695 0696 0697 0698 0699 0700 0701 0702 0703
0704 0705 0706 0707 0708 0709 0710 0711 0712 0713 0714 0715 0716 0717
0718 0719 0720 0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731
0732 0733 0734 0735 0736 0737 0738 0739 0740 0741 0742 0743 0744 0745
0746 0747 0748 0749 0750 0751 0752 0753 0754 0755 0756 0757 0758 0759
0760 0761 0762 0763 0764 0765 0766 0767 0768 0769 0770 0771 0772 0773
0774 0775 0776 0777 0778 0779 0780 0781 0782 0783 0784 0785 0786 0787
0788 0789 0790 0791 0792 0793 0794 0795 0796 0797 0798 0799 0800 0801
0802 0803 0804 0805 0806 0807 0808 0809 0810 0811 0812 0813 0814 0815
0816 0817 0818 0819 0820 0821 082
```

```
96e3 96f4 96g5
2947200500
```

0.583 sec to process 100 student schedules - represented through 64800 clauses and 8000 variables

The regular method took 3 billion nanoseconds (≈ 3 seconds) to process 100 student schedules

Summary

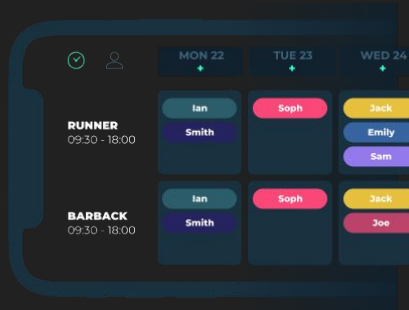
- I was able to successfully create a heuristic-based approach to optimizing schedule development through the use of SAT-solvers.
 - In order to do this I had to encode a system to auto-generate clauses comprised of variables representing a student taking a specific class at a specific time.
 - Further, I had to encode criteria for schedule production.
- I then created a “reader” file which interpreted the large numeric output of SAT4j and converted results back into course-variable notation, laying out the courses that each individual student would be taking.
- Finally, I created a generic algorithm for spacing the schedules proportionally to SAT4j and compared both methods’ runtimes which revealed roughly a 6X speed improvement when utilizing SAT4j.

Conclusion

- Although my initial approach to variable assignment and clause construction was flawed, my revised approach in using variables to represent students, courses, and periods was ultimately successful.
- The regular “brute force” method, although not perfect, worked successfully to create generic schedules and acted as a valuable tool to quantify the improvements obtained by employing SAT4j (and SAT solvers in general).
- The near 6-fold improvement in scheduling execution has significant implications for the benefit of using SAT solvers to other generic algorithms and would likely show only more improvement (in runtime) as initial conditions get more complex and numerous.
- The data proves that my initial hypothesis was accurate.

Real World Application

- In general, the modern problem of creating efficient, accurate schedules is applicable to many aspects of daily life from planning school courses and mapping out work hours, to managing airways/plane takeoff. In terms of course order, scheduling plays an important role for broader institutional effectiveness. Productive scheduling, for example, can boost student retention rates and reduce time to graduation. Ultimately, the heuristic approach to optimizing schedule-creation, depicted through this work, could drastically improve cost-effectiveness and scheduling accuracy.



Errors & Further Study

Errors:

- At first, the student variable wasn't taken into account when expressing the constraints.

Further study:

- With more time to conduct research, I would integrate more features into the scheduler to make it more practical (as we now know it would be applicable).
- Additionally, I would like to test out various different SAT-solvers as they range in quality, and try to identify the most efficient solver for generating schedules.

Citations

- https://www.researchgate.net/publication/268460124_Satisfiability_Methods_for_Colouring_Graphs/fulltext/5721424d08ae0926eb45bd3f/Satisfiability-Methods-for-Colouring-Graphs.pdf
- <https://cse.buffalo.edu/~erdem/cse331/support/sat-solver/index.html>
- <https://www2.cs.sfu.ca/~mitchell/cmpt-827/2015-Fall/Projects/TT-sat-timetable.pdf>
- <https://jix.github.io/varisat/manual/0.2.0/formats/dimacs.html#:~:text=The%20DIMACS%20CNF%20format%20is,a%20negation%20of%20a%20variable.>
- <https://sat4j.gitbooks.io/case-studies/content/running-sat4j-as-a-standalone-solver.html>
- <https://www.borealisai.com/research-blogs/tutorial-9-sat-solvers-i-introduction-and-applications/#:~:text=However%2C%20for%203%2DSAT%20and,millions%20of%20clauses%20in%20practice.>

Thank You For
Listening!