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.

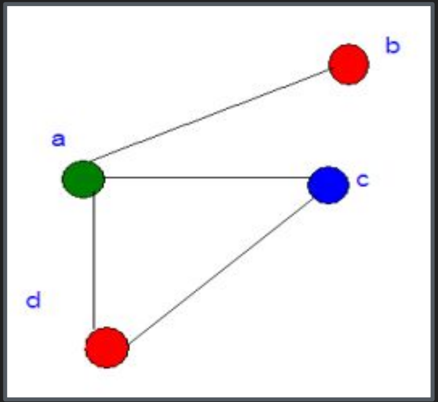
Research

- 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.

$$
\begin{aligned}
&\alpha = P \vee (Q \rightarrow R) && \\
&\neg\alpha = \neg(P \vee (Q \rightarrow R)) && \\
&\equiv \neg(P \vee (\neg Q \vee R)) && \text{Implication law} \\
&\equiv \neg P \wedge (\neg(\neg Q \vee R)) && \text{De-Morgans law} \\
&\equiv \neg P \wedge (Q \wedge \neg R) && \text{De-Morgans law \& Double negation} \\
\therefore\ &\alpha \equiv \neg P \wedge (Q \wedge \neg R) &&
\end{aligned}
$$

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:
    - $(\sim a_1 \lor \sim c_1)$ $(\sim a_2 \lor \sim c_2)$ $(\sim a_3 \lor \sim c_3)$
- Type 2 clause:
    - $(a_1 \lor a_2 \lor a_3)$
- Type 3 clauses:
    - $(\sim a_1 \lor \sim a_2$ $\sim a_1 \lor \sim a_3$ $\sim a_3 \lor \sim a_2)$

Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses
1	~a1v~c1	8	~d2v~c2	15	c1vc2vc3	22	~b2v~b3
2	~a2v~c2	9	~d1v~c1	16	d1vd2vd3	23	~c1v~c2
3	~a3v~c3	10	~a1v~b1	17	~a1v~a2	24	~c1v~c3
4	~a1v~d1	11	~a2v~b2	18	~a1v~a3	25	~c2v~c3
5	~a2v~d2	12	~a3v~b3	19	~a3v~a2	26	~d1v~d2
6	~a3v~d3	13	a1va2va3	20	~b1v~b2	27	~d1v~d3
7	~d3v~c3	14	b1vb2vb3	21	~b1v~b3	28	~d2v~d3

Background & Expected Outcome

As an example the formula (x ∨ y ∨ ¬z) ∧ (¬y ∨ z) could be encoded as this:

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

- 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 heighted runtime which would sort faster and therefore be more cost-efficient and viable.

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)

```java
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);
```

```java
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] + " ");
        }
    }
}
```

```java
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

```java
//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

```
p cnf 400 3240
-1 -9 0
-1 -17 0
-1 -25 0
-1 -33 0
-1 -41 0
-1 -49 0
-1 -57 0
-1 -65 0
-1 -73 0
-9 -17 0
-9 -25 0
-9 -33 0
-9 -41 0
-9 -49 0
-9 -57 0
-9 -65 0
-9 -73 0
-17 -25 0
-17 -33 0
-17 -41 0
-17 -49 0
-17 -57 0
-17 -65 0
-17 -73 0
-25 -33 0
-25 -41 0
-25 -49 0
-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”

```java
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] = "";
```

```java
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);
```

```java
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";
    }
```

```java
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.OriginalWLClause => 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
```

```
0i3 90j4 91a5 91b6 91c7 91e1 91f2 91g3 91h4 91i5 91j6 92a
95g3 95h4 95i5 95j6 96a7 96c1 96d2 96e3 96f4 96g5 96h6
100d2 100e3 100f4 100g5 100h6 100i7 3034063700
```

```
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

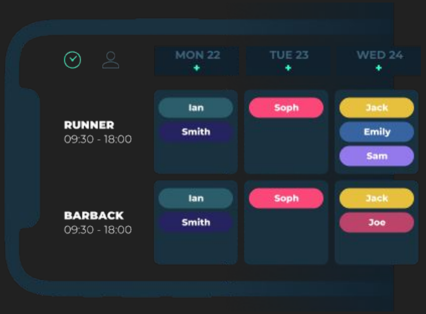
- I was able to successfully create a heuristic-based approach to optimizing schedule development through the use of SAT-solvers.
    - In order to do this I had to encode a system to auto-generate clauses comprised of variables representing a student taking a specific class at a specific time.
    - 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!