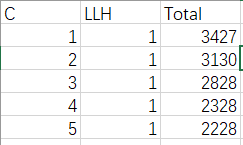
COMP5920 Scheduling Coursework 2

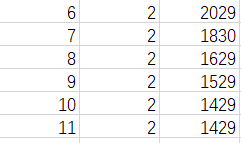
Meng Fanhui (sc19fm) 201373470

1. For this experiment, I plan to run each low-level heuristic component several times, until no obvious reduction of the solution cost then run the next LLH.

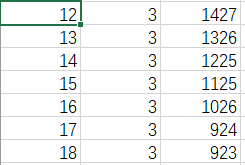
For the chart below, C represent the total number of LLH activations, LLH represent which LLH is executed, total represent total constraint violations.



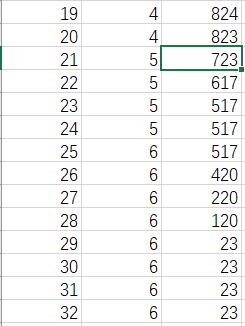
From the first to the 5th attempts, I run LLH 1, and the total constraint violations drops from initial 4027 to 2228. And I found there are only small reduce between 4th and 5th attempts. Therefore, I think LLH 1 couldn’t make the solution much better at this time, then I run LLH 2 for the next time.



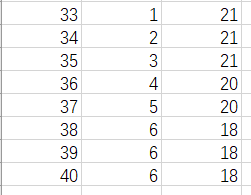
For LLH2, it not make the solution much better, and for 10th and 11th attempts, the soultion cost remain the same. So I move on to LLH3.



For LLH3, each attempt only reduce about 100 to the solution cost, and I think this is acceptable, because the number is down to 1000, so it can’t change too much. When the reduction down to single digits, move on to the LLH4.

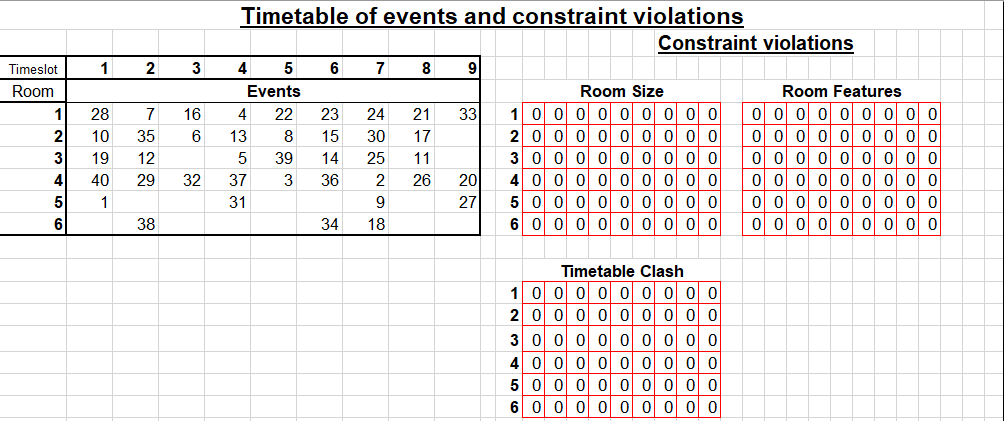


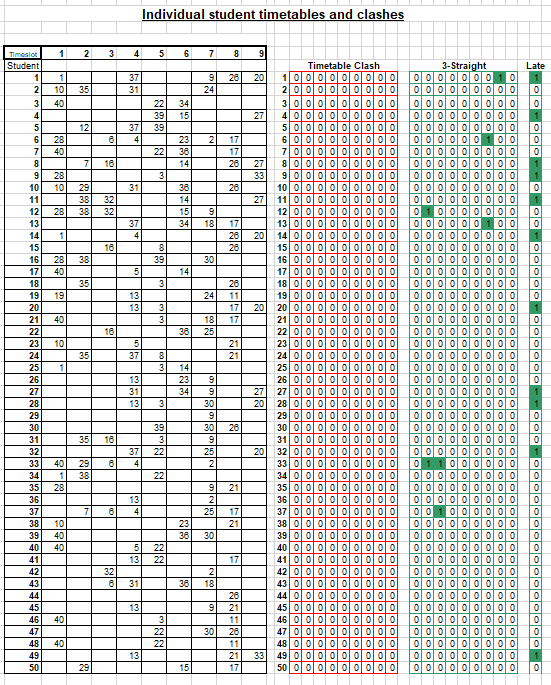
Then I do the same thing for the next few experiments. When it comes to small reduction, go for the next LLH algroithm. Finally, there are remarkable decrease when I run LLH 6. The total constraint violations drops down to 23. And I do another few tries until the number is no longer change.

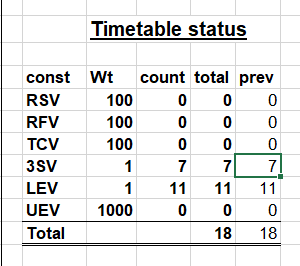


After another round of attempts, this solution’s final total constraint violations remains 18, which is fairly good. Because there is no hard constraint violations in the timetable of event as well as individual student timetable. Thus, this solution satisfy every requirement (e.g. room size, room features, no clash in student timetable). All of these 18 constraint violations are in green box, which means in the student timetable there are three successive event violations and late event violations. And these latter later two are highlighted in green, as they are soft constraints, which are not as serious as hard constrints. Basically, the experiment with my strategy get a reasonably good solution.

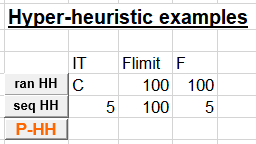
The screenshots of the final solution timetable and timetable status show as below.



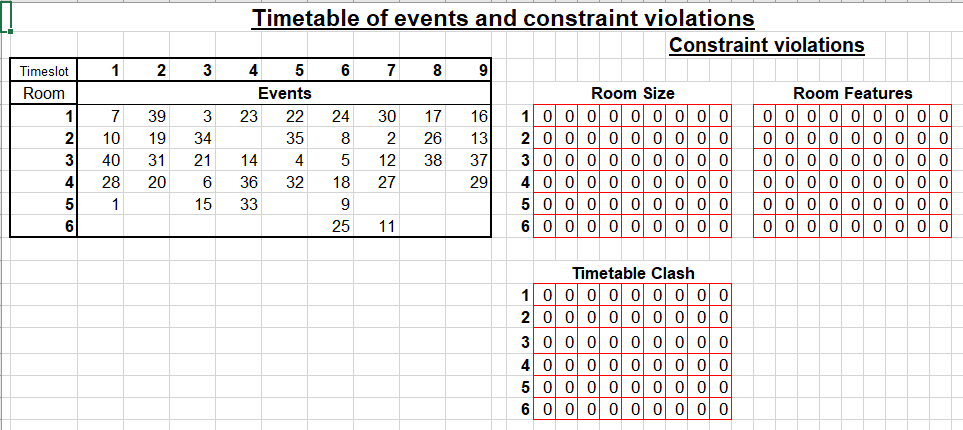


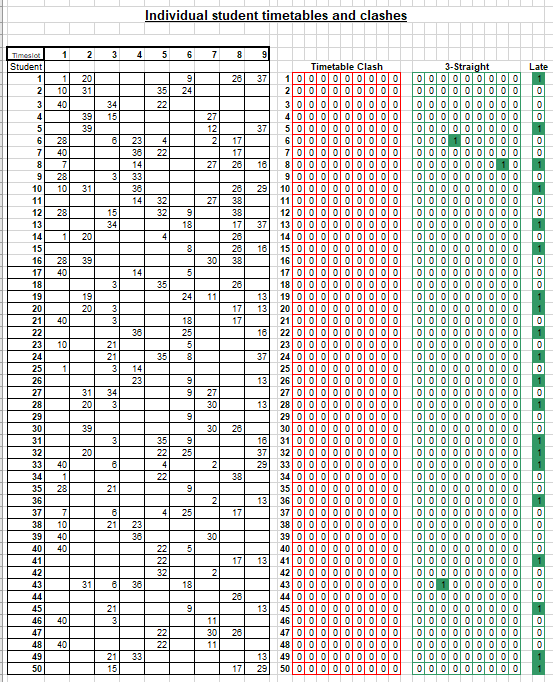


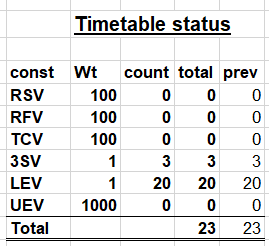
1. Change IT parameter value to C



Screenshots show as below.







1. Pseudocode of my P-HH shows as below.

public class pseudocode{

    //define counter for each LLH activations and the total activation

    public static int counter = 0;

    public static int counter1 = 0;

    public static int counter2 = 0;

    public static int counter3 = 0;

    public static int counter4 = 0;

    public static int counter5 = 0;

public static int counter6 = 0;

//define the performance after each LLH

public static long total = 0;

public static long total1 = 0;

public static long total2 = 0;

public static long total3 = 0;

public static long total4 = 0;

public static long total5 = 0;

public static long total6 = 0;

//method for initialize

    public static void initialize(int[] ranking){

        //Run each LLH independently for 10 times

        for(int i = 0; i < 10; i++){

            call LLH1;

        }

        //get the total constraint violations after 10 times LLH1

        total1 = getTotal();

        //numerical the performance of LLH1

        total1 = (total1 / 4027 /10) \* 100;

        //put it into the array, storing the ranking

        ranking.append(total1);

        //restart, prepare for the next LLH

        restart();

        //for LLH2

        for(int i = 0; i < 10; i++){

            call LLH2;

        }

        total2 = getTotal();

        total2 = (total2 / 4027 /10) \* 100;

        ranking.append(total2);

        restart();

        //I wouldn't write all of them, cause it's the same for the  //rest LLHs initialization

    }

    //find the minimum, which is the best performance in the ranking  //array

    public static int findMin(int [] a){

        int min;

        int tmp = a[0];

        for(int i = 0; i < length.a - 1; i++){

            if(a[i] < tmp){

                min = i;

                tmp = a[i];

            }

        }

        return min;

    }

    //For those LLH who has not been selected for a while

    public static void rarely(){

        //if 10 attempts and none of them are LLH1, then assign 15  //weight to LLH.

        if ((counter1 / counter) < 0.1){

            total1 += 15;

        }

        else if ((counter2 / counter) < 0.1){

            total1 += 15;

        }

       //I'm not gonna write all of it in here, cause they all the same

}

//restart method

    public static void restart(){

}

    public static void main(String[] args){

        //array for storing ranking

        int [] ranking;

        //initialize

        initialize(ranking);

        //drop out of loop, until total constraint violation is smaller  //than 50

        do{

            //update total constraint violations

            int total = getTotal();

            //For those LLH who has not been selected for a while

            rarely();

            //find the minimum, which is the best performance in the  //ranking array

            findMin(ranking);

            switch(min){

                case 0:

                    call LLH1;

                    //update it's performance

                    //to judge it's performance is to use the total1  //(newly total constraint violations) divide by the//previous total

                    //then multiply 100 to make it as integer,  //then replace the corresponding number in the rank//ing array

                    total1 = (total1 / total) \* 100;

                    ranking[0] = total1;

                    //update counter for LLH1

                    counter1++;

                    //update counter for all attempts

                    counter++;

                    break;

                case 1:

                    call LLH2;

                    //update it's performance

                    total2 = (total2 / total) \* 100;

                    ranking[1] = total2;

                    counter2++;

                    counter++;

                    break;

                case 2:

                //the same as above

                case 3:

                //the same as above

                case 4:

                //the same as above

                case 5:

                //the same as above

            }

        }while(total > 50);

    }

}