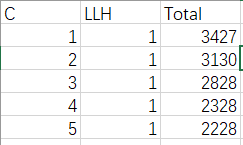
**COMP5920 Scheduling Coursework 2**

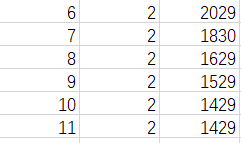
Meng Fanhui (sc19fm) 201373470

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

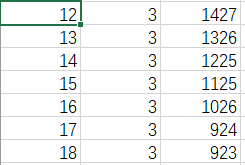
For the chart below, C represents the total number of LLH activations, LLH represents which LLH is executed, "total" represents the total constraint violations.



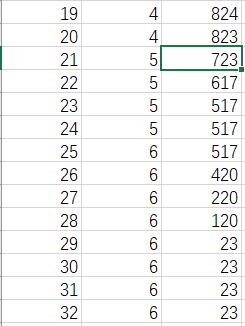
From the first to the 5th attempts, I run LLH 1, and the total constraint violations drop from initial 4027 to 2228. And I found there are only a small reduce between 4th and 5th attempts. Therefore, I think LLH 1 couldn't make the solution much better at this time. Then I run LLH 2 for the next time.



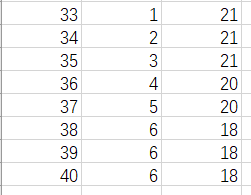
For LLH2, it does not make the solution much better, and for 10th and 11th attempts, the solution cost remain the same. So I move on to LLH3.



For LLH3, each attempt only reduces 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.

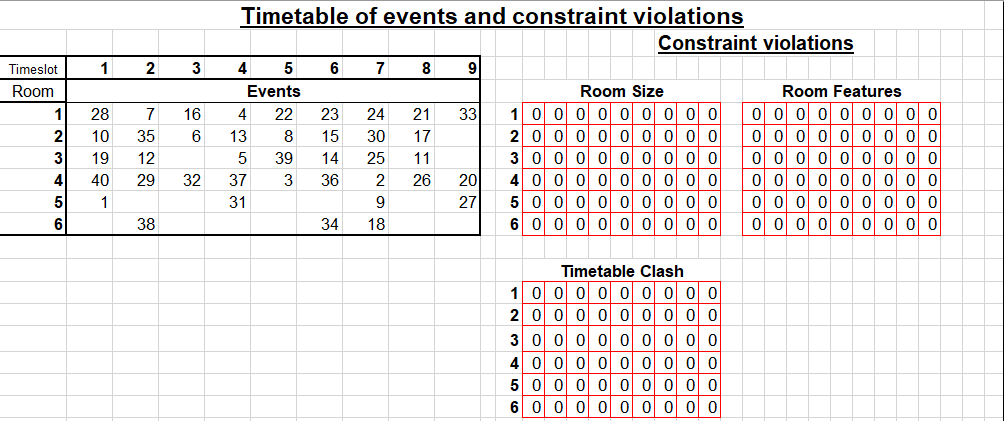


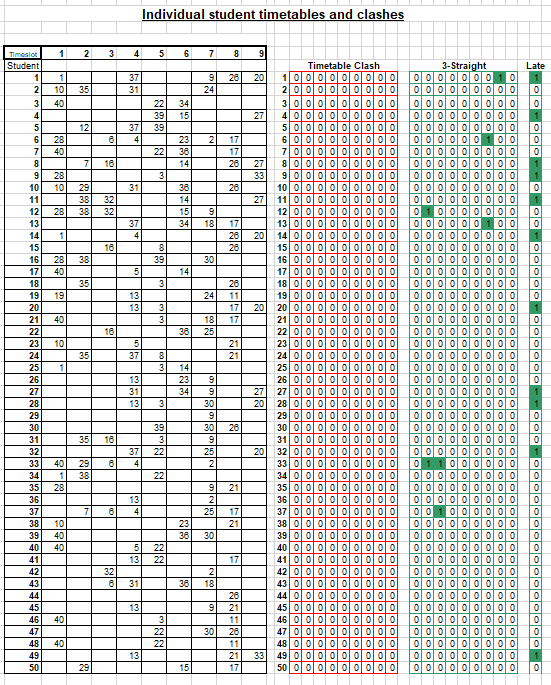
Then I do the same thing for the next few experiments. When it comes to small reduction, go for the following LLH algorithm. Finally, there is a remarkable decrease when I run LLH 6. The total constraint violations drop down to 23. And I do another few tries until the number is no longer change.

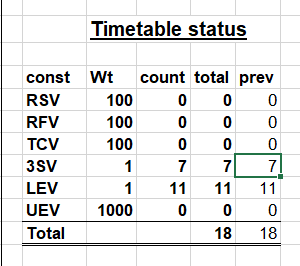


After another round of attempts, this solution's final total constraint violations remains 18, which is relatively good. Because there is no hard constraint violations in the timetable of the event as well as individual student timetable, thus, this solution satisfies every requirement (e.g. room size,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

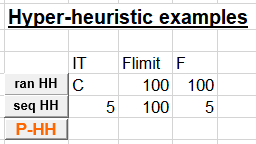
The screenshots of the final solution timetable and timetable status show as below.



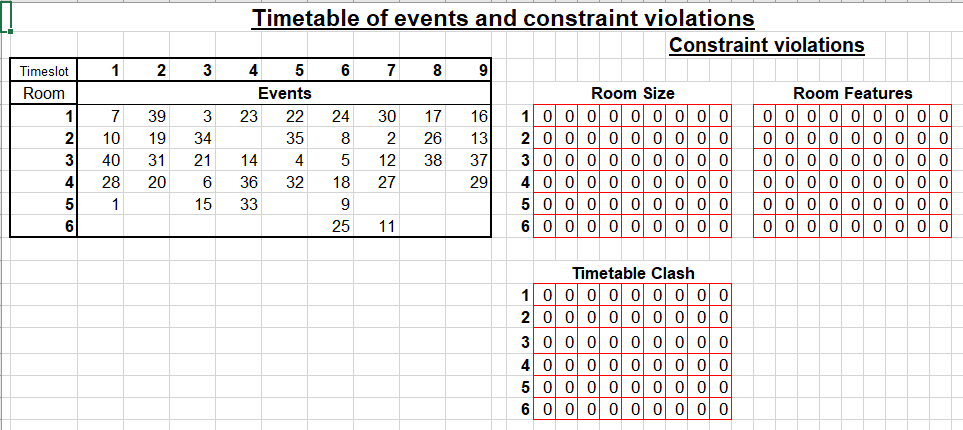


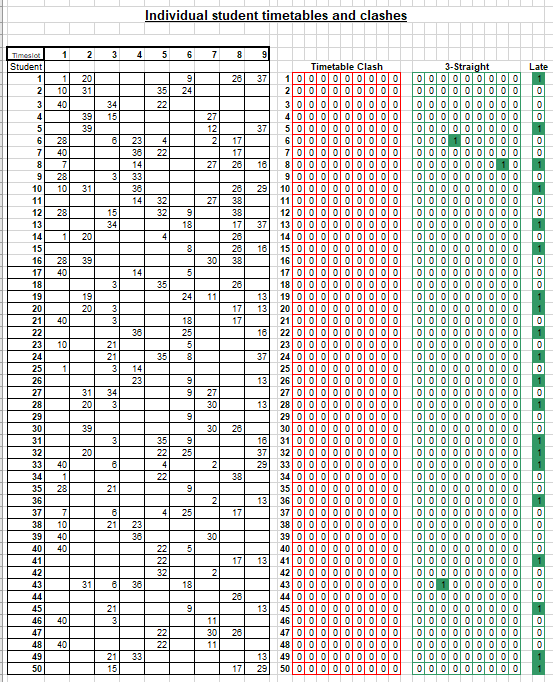


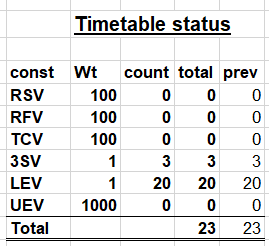
1. Change IT parameter value to C



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;

    //method for initialize

    public static void initialize(int[] ranking){

        //Run each LLH independently for 10 times

        total = getTotal;

        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

        //Using the total constraint violations at the very begining  //minus the current violations

        total1 = total - total1;

        //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 = total - total1;

        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 findMax(int [] a){

        int max;

        int tmp = a[0];

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

            if(a[i] > tmp){

                max = i;

                tmp = a[i];

            }

        }

        return max;

    }

    //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 it.

        if ((counter1 / counter) < 0.1){

            total1 += 50;

        }

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

            total1 += 50;

        }

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

    }

public static void restart(){

    }

    public static void main(String[] args){

        //array for storing ranking

        int [] ranking;

        //initialize

        initialize(ranking);

        //drop out of the 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 max, which is the best performance in the  //ranking array

            findMax(ranking);

            switch(max){

                case 0:

                    call LLH1;

                    //update it's performance

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

                   //then replace the corresponding number in the  //ranking array

                    total1 = total - total1;

                    ranking[0] = total1;

                    //update counter for LLH1

                    counter1++;

                    //update counter for all attempts

                    counter++;

                    break;

                case 1:

                    call LLH2;

                    //update it's performance

                    total2 = total – total2;

                    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

            }

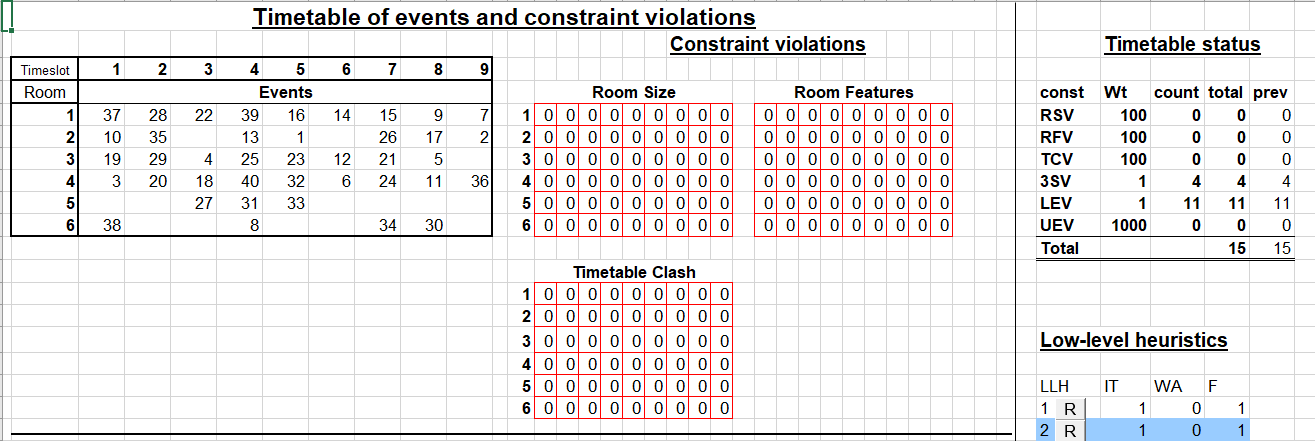
        }while(total > 20);

    }

}

1. Please see the Excel file and the log.txt file, which is my P-HH log file.
2. With my strategy for the first time, it took me 40 times of LLH activations to find the best solution, which the violations are 18.

The result of my P-HH experiment: it took about 5 minutes and 50 attempts to find out the initialization of each LLH's performance. And it took another 3 minutes and 42 attempts to reach the best solution, which the violations are 15.



As for random LLH, the performance of this hyper-heuristic is all about luck. I did some experiments on it, and the result can be very different. One experiment took me 10 minutes, and the final total constraint violations are 113. Another experiment also took about 10 minutes, but it reaches 17 of the violation at the first 3 minutes. In conclusion, this algorithm picks the random LLH for no reason so that it can be the worst or sometimes the best algorithm.

All in all, find and use the best performance LLH can be the most efficient way of finding the best solution. But the performance is not fixed, so keep update the performance and use the best performance at a specific time is always the best choice. Perhaps, it's not the most efficient way when it compares with other methods. Like when I do the first experiment, the attempts are less than the P-HH attempts to find the best solution. However, it's stable. It can always find a suitable solution for a short period.