Module code: CS4402
Matriculation number: 200017941

Practical 1: The Operations Room Report

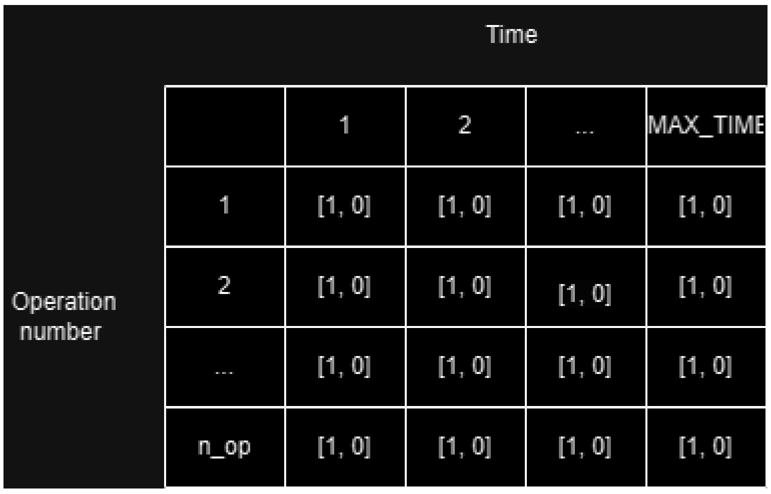
Overview

For this Constraint Satisfactory Problem, I have provided two Constraint Models: *model.eprime* and *model2.eprime*. Both these models have a different viewpoint: *Model.eprime* uses explicit representation and *model2.eprime* uses occurrence representation. The reason why I have two models is because the initial model that I designed (*model2.eprime*), didn't perform as well as I would like it to. As a result, I decided to design another model (*model.eprime*) with the aim of addressing some of the issues that were faced with the first model. This report will document both these models with the aim of providing some discussion and insight into what some of the issues were with the first model and the changes I made with my approach in designing the second model. However, *model.eprime* will serve as my main submission and so only the performance of this model will be evaluated.

Design of model2.eprime

<u>Variables</u>

There are two decision variables that have been employed for this Constraint Model: *timetable* and *total_time*. The first variable is a nested data structure where it is a sequence of sets. In this 2-dimensional array the columns represent the time, and the rows represent the operation number, therefore it stores the start and end times for each operation and serves as the main decision variable. The second variable holds an integer value which represents the time taken to complete all the operations. This is part of the optimization function where the time taken needs to be minimized. Below is a diagrammatic representation of *timetable*.



		Time			
		1	2	...	MAX_TIME
Operation number	1	[1, 0]	[1, 0]	[1, 0]	[1, 0]
	2	[1, 0]	[1, 0]	[1, 0]	[1, 0]
	...	[1, 0]	[1, 0]	[1, 0]	[1, 0]
	n_op	[1, 0]	[1, 0]	[1, 0]	[1, 0]

Domains

For *timetable*, the domain of values consists of 1s and 0s since it is an occurrence representation. Whereas for *total_time*, the domain of values are integers ranging from 1 to 1000. This is the upper bound because total_time is the variable being optimized and therefore can take any value depending on the parameters provided. Such a large number was chosen so that the model can schedule really large amounts of operations without running out of space.

Constraints

The following set of constraints ensure that each operation is scheduled appropriately:

- *total_time <= sum([duration[index] | index : int(1..n_op)])*

  This constraint ensures that the total time taken to complete all the operations is less than or equal to the worst-case scenario where all the operations are done one after the other. The duration for each of the operations, which is stored in the *duration* array, are added and compared against *time_taken*.

-
  ```
  forAll col : int(1..sum(duration)) .
      (col < total_time) ->
        sum([timetable[row, col] | row : int(1..n_op)]) >= 1
  ```

  This constraint makes sure that all the operations are performed before the end time of the last operation which is indicated by the *total_time* variable. The *time-taken* variable acts as a marker which is used in the conditional to make sure that the sum of the 1s in each column that come before the marker is greater than or equal to 1. This is because each 1 represents an operation currently being carried out and therefore if each time slot has one or more 1s, then it is being optimized.

-
  ```
  forAll col : int(1..MAX_TIME) .
      (col >= total_time) ->
        sum([timetable[row, col] | row : int(1..n_op)]) = 0
  ```

  This constraint is similar to the one above except it makes sure that all the time slots after the last operation has finished are empty. This is important since *total_time* represents the end time of the last operation which needs to be minimized. As a result, all the operations are pushed to the left as much as possible.

-
  ```
  forAll row : int(1..n_op) .
      sum([timetable[row, col] | col : int(1..sum(duration))]) = duration[row]
  ```

This constraint ensures that each operation is completed within the given duration. This is done by counting the number of 1s in each row and equating it to the value in the *duration* array for the current operation.

-
  ```
  forAll row : int(1..n_op) .
      forAll col : int(1..sum(duration)-1) .
          (timetable[row, col] = 1 /\ timetable[row, col+1] = 0) ->
              forAll num : int(col+1..sum(duration)) .
                  timetable[row, num] = 0
  ```

  This constraint makes sure that all the 1s in each row are next to each other since each operation needs to be completed in one go without any breaks in between. This is done using the assumption that whenever there is a 1 followed by a 0, this marks the end of an operation, so all the timeslots after that point are equated to 0 for each row. If there are several occurrences of a 1 followed by a 0 for each row, this means that there are breaks within the operation which isn't allowed. But 0s that come before a 1 is allowed.

-
  ```
  forAll col : int(1..sum(duration)) .
      sum([timetable[row, col] | row : int(1..n_op)]) <= n_surgeons
  ```

  This constraint ensures that for every time slot, the number of surgeons in charge is less than or equal to the total number of available surgeons. This is done by adding up all the 1s in each column and comparing it with the number of surgeons available since each operation only needs one surgeon to be in charge.

-
  ```
  forAll col : int(1..sum(duration)) .
      forAll staff : int(1..n_specialities) .
          sum([manning[row, staff] * timetable[row, col] | row : int(1..n_op)]) <=
  available_personnel[staff]
  ```

  This constraint ensures that for every time slot, the number of specialists that are operating is less than or equal to the total number of available specialists. For each time slot, each row is checked to see if there are any operations currently taking place. When a 1 is encountered then the operation number for that row is used as the index for the *manning* array which stores the required number of specialists for each operation. The sum for each time slot across all the currently ongoing operations is calculated and compared against the total number of available specialists.

-
  ```
  forAll pair : int(1..n_ordering_relations) .
      forAll col  : int(1..sum(duration)) .
          timetable[order[pair, 1], col] * timetable[order[pair, 2], col] = 0
  ```

This constraint is used to make sure that if there is an order imposed between two operations, then there cannot be any overlap. This is done by taking the row of the first operation and the row of the second operation and then applying an AND operation. The results should be all 0s, which indicates no overlap.

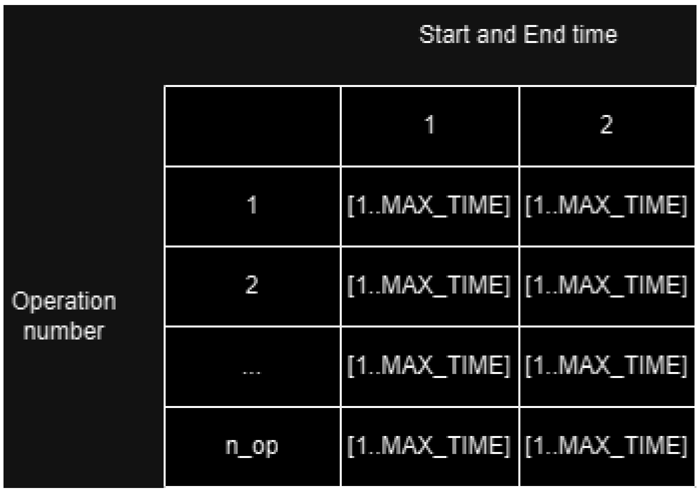
-
```
forAll pair : int(1..n_ordering_relations) .
    forAll col : int(1..sum(duration)) .
        timetable[order[pair, 1], col] = 1 ->
        (sum([timetable[order[pair, 2], num] | num : int(1..col)]) = 0)
```

This constraint ensures that the given order of operations is being followed. Given a pair of operations from the *order* array which states the order as a series of tuples, it goes through each time slot in each row. When a 1 is detected in the row of the first operation in the tuple, all the columns for the row of the second operation in the tuple are equated to 0. This condition ends after all the timeslots for the first operation in the tuple are 0, which means that the second operation in the tuple is free to take place.

Design of model.eprime

<u>Variables</u>

This Constraint Model makes use of three decision variables: *op_schedule*, *timetable* and *total_time*. The first variable is a nested data structure where it is a set of sets. The columns represent the start and end times, and each row is the operation number. This is the main decision variable for the model as it stores the timings for each operation. The other two variables are the same as before, except this time *timetable* is being used as an auxiliary variable. Below is a diagrammatic representation of *op_schedule*.



		Start and End time	
		1	2
Operation number	1	[1..MAX_TIME]	[1..MAX_TIME]
	2	[1..MAX_TIME]	[1..MAX_TIME]
	...	[1..MAX_TIME]	[1..MAX_TIME]
	n_op	[1..MAX_TIME]	[1..MAX_TIME]

Domains

For *op_schedule,* the domain of values consists of integers ranging from 1 to 1000. The *total_time* is given a maximum value of 1000, and *op_schedule* must be consistent with this since its row contains values for the operation time. The other two domains of the other two variables are the same as described above.

Constraints

The following set of constraints ensure that each operation is scheduled appropriately:

- *max([op_schedule[op, 2] | op : int(1..n_op)]) = total_time*

  This constraint makes sure that the end time of the last operation is equal to the *total_time* variable. This is done by finding the largest value stored in the second column of the *op_schedule* array. As a result, the optimization function is linked to the main decision variable.

- *forAll op : int(1..n_op) .*
  op_schedule[op, 2] - op_schedule[op, 1] = duration[op]

  This constraint ensures that the start and end time of each operation follows the given duration. This is done by simply subtracting the two values in each of the columns and equating it to the value stored in *duration* for each row.

- *forAll pair : int(1..n_ordering_relations) .*
  op_schedule[order[pair, 1], 2] <= op_schedule[order[pair, 2], 1]

  This constraint ensures that the specified ordering of operations is being followed. This is done by specifying the end time of the first operation in the tuple to be less than or equal to the start time of the second operation in the tuple for each row in the *order* array.

- *forAll op : int(1..n_op) .*
  forAll time : int(1..sum(duration)) .
  time >= op_schedule[op, 1] /\ time < op_schedule[op, 2] -> timetable[op, time] = 1

  This is the channeling constraint, and it makes sure that the values in *op_schedule* and *timetable* decision variables are consistent. For each operation it goes through all the time slots and checks if the current time slot from the *timetable* variable is between the start and end time of the current operation from the *op_schedule* variable. If it is, then a 1 is added to the *timetable* variable using the time slot and operation number as the indices.

- *forAll op : int(1..n_op) .*
  forAll time : int(1..sum(duration)) .
  time < op_schedule[op, 1] /\ time >= op_schedule[op, 2] -> timetable[op, time] = 0

  This is also a channeling constraint to make sure that both variables are consistent. This time instead of adding 1s to the appropriate elements in the *timetable* variable, it adds 0s to the remaining elements where there aren't any operations taking place. This establishes a strict connection between the two variables.

- *forAll col : int(1..sum(duration)) .*
  sum([timetable[row, col] | row : int(1..n_op)]) <= n_surgeons

  This constraint makes sure that the number of surgeons in charge at any given time isn't over the number of available surgeons. This constraint is defined in the same way as the previous model since it uses the *timetable* auxiliary variable.

- *forAll col : int(1..sum(duration)) .*
  forAll staff : int(1..n_specialities) .
  *sum([manning[row, staff] * timetable[row, col] | row : int(1..n_op)]) <= available_personnel[staff]*

  This constraint makes sure that the number of specialists that are currently operating doesn't exceed the total number of available specialists. Again, this constraint is defined in the same way as the previous model since it uses the *timetable* auxiliary variable.

Comparing model2.eprime and model.eprime

For *model2.eprime*, I chose to go with a 2-dimensional array which uses occurrence representation because it was the most intuitive way of thinking about modelling this problem. Since it is a scheduling problem, time is obviously a key component. For this category of problems, time is often represented as an ordered sequence, which in this case is the columns. And since for each time-period an operation is assigned, naturally the rows represent the set of operations that can take place at any given time. This sort of table structure acts as a timetable (hence the name) which makes it relatively straightforward to implement certain constraints such as the ones for ensuring that the available surgeons and specialists limit is not exceeded. Additionally, the order of operations is easy to visualize, and overlaps are easily identified. However, most of the other constraints which include incorporating the optimization function, ensuring the correct order and making sure that the 1s are contiguous are non-trivial. It led to lengthy constraints and extra constraints that had to be added. This made for an inelegant model especially in terms of its efficiency. The time taken to produce a solution is longer than it should be and for most of the parameter files ranging from medium to very large, it was unsuccessful in generating a solution. This was mainly due to most of the constraints using loops with considerably large intervals.

Due to all these drawbacks I decided to design an alternative Constraint Model which doesn't use an occurrence representation for all the constraints. So, for *model.eprime*, I chose to use a 2-dimensinoal data structure that uses explicit representation instead. Alongside this, I also made use of the auxiliary *timetable* variable. This meant that for constraints that were easily defined using an explicit representation I made use of *op_schedule* and for constraints that were easier with an occurrence representation, *timetable* was used. This addressed the issue I was having with certain constraints being unnecessarily complicated to handle such as the ones for ensuring the right order. The only additional constraint that was required was the channeling constraint to ensure that both variables stayed consistent. Additionally, the solution produced was a lot easier to assess since the start and end times are explicitly stated. The overall model appears to be more concise with fewer and simpler constraints compared to the first model. This also resulted in solutions being generated faster since loops weren't used as much in the constraints, and a lot more of the provided instances were solvable.

Evaluation

When developing this model, I made sure to test the output of each constraint individually, to verify that it performs as it should. Then more constraints were added incrementally to test the output produced with all the constraints in place. Doing it this way allowed me to pinpoint exactly which constraint is the source of any errors that occurred.

The model was tested with the provided instances and all the output files that were generated are stored in separate directories. For *model.eprime*, the solutions are in the *model_solutions* directory, where it has sub-directories for each input size which corresponds to the *instances* directory. The same was done for *model2_eprime*, where the solutions are in the *model2_solutions* directory. As I mentioned in the overview, this section will only focus on the solutions generated by *model.eprime* since this model has proven to solve a much larger set of the instances and at a much more reasonable time.

My model is able to solve all the small and medium instances without any issues. For the large instances, it was able to solve almost all the instances except for two: *3.param* and *6.param*. Most of the very large instances were unable to be solved except for one: *1.param*. For the small and medium instances, the solution seems to be provided reasonably quickly (roughly 0.01-0.0001 for small instances and 0.1-0.01 for medium instances), whereas for the large instances it took a noticeably longer time (roughly 1-0.1). This is consistent with the number of nodes that were visited for each instance. For small and medium, the number of nodes is relatively smaller (roughly 100-1000 nodes for small instances and 1000-6000 for medium instances), whereas for large instances the number of nodes is considerably larger (roughly 10000-20000 nodes). However, there are a few exceptions where a few instances took unexpectedly long to solve and have a much larger number of nodes, for example medium *3.param*. All the solutions that were given for these instances seem to be correct for the most part as all the constraints were being obeyed. The time assigned to each operation is done in such a way that the specified order is followed and the number of surgeons and specialists operating at each time slot doesn't exceed the provided limit. More importantly, the total time taken is being optimized to be as short as possible. However, a key observation from the *.info* file showed multiple solutions being found for some of the instances. This potentially indicates symmetries in the solutions which weren't taking into consideration as I haven't added a constraint for handling symmetries.

For the instances that it couldn't solve, it generated *.minion* and *.MINIONSTATS* files which indicate that the solver is able to parse the Constraint Model and process it to produce a solution but is unable to reach a final solution. This means that there is a semantical issue with how I designed the model instead of a syntactical error. I'm assuming that for really large instances, due to the size of the parameters, my model was unable to efficiently narrow down the search space. Below, I have provided the solutions generated for all the successful instances as well as the key information about the model's performance which was included in the *.info* file. As part of the solution, the following two decision variables are provided: *total_time* which states the end time of the last operation and *op_schedule* which has the start and end time of each operation in the same order as the *duration* array. Note that the model was tested using Minion since only the default Savile Row settings were used and all the information below is included in the *model_solutions* directory.

<u>Small</u>

- *1.param*:
  total_time = 9
  op_schedule = [[1, 1], [1, 5], [5, 7], [5, 9], [7, 8], [9, 9]]

  SavileRowTotalTime = 0.319
  SolverTotalTime = 0.001023
  SolverSolveTime = 0.000189
  SolverNodes = 46
  SolverSatisfiable = 1
  SolverSolutionsFound = 1

- *2.param*:
  total_time = 17
  op_schedule = [[1, 1], [1, 4], [1, 2], [4, 7], [4, 8], [7, 9], [8, 11], [11, 15], [15, 17], [15, 16], [9, 15], [17, 17]]
  SavileRowTotalTime = 0.614
  SolverTotalTime = 0.009182
  SolverSolveTime = 0.00166
  SolverNodes = 258
  SolverSatisfiable = 1

SolverSolutionsFound = 1

- *3.param*:
  total_time = 16
  op_schedule = [[1, 1], [1, 3], [3, 5], [1, 6], [6, 10], [5, 7], [7, 8], [10, 12], [8, 9], [9, 15], [12, 16], [16, 16]]

  SavileRowTotalTime = 0.615
  SolverTotalTime = 0.007858
  SolverSolveTime = 0.000907
  SolverNodes = 302
  SolverSatisfiable = 1
  SolverSolutionsFound = 1

- *4.param*:
  total_time = 17
  op_schedule = [[1, 1], [6, 11], [1, 3], [3, 6], [3, 7], [6, 9], [13, 16], [7, 13], [16, 17], [13, 16], [11, 12], [17, 17]]

  SavileRowTotalTime = 0.624
  SolverTotalTime = 0.016198
  SolverSolveTime = 0.005328
  SolverNodes = 956
  SolverSatisfiable = 1
  SolverSolutionsFound = 4

- *5.param*:
  total_time = 14
  op_schedule = [[1, 1], [1, 5], [1, 2], [2, 3], [5, 8], [5, 8], [8, 10], [5, 6], [10, 12], [10, 11], [12, 14], [14, 14]]

  SavileRowTotalTime = 0.616
  SolverTotalTime = 0.011845
  SolverSolveTime = 0.004127
  SolverNodes = 352
  SolverSatisfiable = 1
  SolverSolutionsFound = 1

- *6.param*:
  total_time = 27
  op_schedule = [[1, 1], [1, 10], [1, 3], [1, 8], [14, 20], [3, 4], [10, 14], [14, 18], [10, 13], [18, 22], [20, 27], [27, 27]]

SavileRowTotalTime = 1.008
SolverTotalTime = 0.017953
SolverSolveTime = 0.004333
SolverNodes = 866
SolverSatisfiable = 1
SolverSolutionsFound = 2

- *7.param*:
  total_time = 19
  op_schedule = [[1, 1], [3, 5], [1, 3], [5, 7], [3, 5], [5, 9], [7, 8], [9, 10], [9, 15], [15, 19], [8, 13], [19, 19]]

  SavileRowTotalTime = 0.622
  SolverTotalTime = 0.013681
  SolverSolveTime = 0.002971
  SolverNodes = 490
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

- *8.param*:
  total_time = 19
  op_schedule = [[1, 1], [1, 4], [4, 5], [5, 7], [5, 6], [7, 10], [10, 14], [14, 15], [15, 17], [15, 16], [16, 19], [19, 19]]

  SavileRowTotalTime = 0.591
  SolverTotalTime = 0.019086
  SolverSolveTime = 0.010657
  SolverNodes = 562
  SolverSatisfiable = 1
  SolverSolutionsFound = 1

- *9.param*:
  total_time = 21
  op_schedule = [[1, 1], [1, 2], [2, 6], [6, 7], [2, 5], [9, 12], [12, 14], [5, 9], [12, 19], [19, 21], [14, 20], [21, 21]]

  SavileRowTotalTime = 0.651
  SolverTotalTime = 0.015395
  SolverSolveTime = 0.003883
  SolverNodes = 500
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

- *10.param*:
  total_time = 11
  op_schedule = [[1, 1], [1, 5], [5, 7], [1, 5], [5, 6], [6, 8], [8, 10], [7, 8], [8, 9], [9, 11], [10, 11], [11, 11]]

  SavileRowTotalTime = 0.586
  SolverTotalTime = 0.009853
  SolverSolveTime = 0.001259
  SolverNodes = 299
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

<u>Medium</u>

- *1.param*:
  total_time = 28
  op_schedule = [[1, 1], [6, 9], [1, 2], [2, 6], [2, 6], [6, 11], [11, 13], [16, 17], [9, 11], [11, 16], [16, 21], [2, 3], [13, 15], [17, 19], [21, 24], [24, 28], [19, 22], [28, 28]]

  SavileRowTotalTime = 1.012
  SolverTotalTime = 0.039812
  SolverSolveTime = 0.011451
  SolverNodes = 1705
  SolverSatisfiable = 1
  SolverSolutionsFound = 3

- *2.param*:
  total_time = 32
  op_schedule = [[1, 1], [1, 2], [2, 3], [1, 2], [3, 4], [5, 7], [4, 6], [2, 5], [5, 12], [12, 19], [7, 11], [12, 14], [7, 9], [19, 26], [26, 32], [26, 29], [19, 26], [32, 32]]

  SavileRowTotalTime = 1.039
  SolverTotalTime = 0.051491
  SolverSolveTime = 0.015336
  SolverNodes = 1960
  SolverSatisfiable = 1
  SolverSolutionsFound = 3

- *3.param*:
  total_time = 35
  op_schedule = [[1, 1], [1, 2], [2, 4], [2, 5], [5, 8], [8, 11], [5, 14], [14, 16], [16, 17], [17, 18], [18, 20], [21, 27], [20, 21], [16, 22], [24, 27], [21, 24], [27, 35], [35, 35]]

SavileRowTotalTime = 1.037
SolverTotalTime = 9.14905
SolverSolveTime = 9.11387
SolverNodes = 150531
SolverSatisfiable = 1
SolverSolutionsFound = 4

- *4.param*:
  total_time = 25
  op_schedule = [[1, 1], [1, 2], [10, 11], [2, 3], [3, 4], [19, 21], [3, 4], [4, 9], [9, 10], [10, 12], [10, 11], [11, 14], [14, 19], [14, 15], [15, 23], [21, 25], [23, 25], [25, 25]]

  SavileRowTotalTime = 0.921
  SolverTotalTime = 0.105283
  SolverSolveTime = 0.080821
  SolverNodes = 2730
  SolverSatisfiable = 1
  SolverSolutionsFound = 4

- *5.param*:
  total_time = 28
  op_schedule = [[1, 1], [1, 6], [6, 7], [1, 4], [6, 7], [7, 10], [10, 12], [12, 15], [15, 16], [7, 9], [9, 12], [17, 23], [10, 11], [15, 17], [23, 28], [23, 25], [12, 15], [28, 28]]

  SavileRowTotalTime = 0.939
  SolverTotalTime = 0.045697
  SolverSolveTime = 0.018436
  SolverNodes = 1117
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

- *6.param*:
  total_time = 23
  op_schedule = [[1, 1], [1, 4], [1, 8], [4, 5], [5, 6], [8, 9], [9, 10], [10, 13], [13, 19], [5, 6], [9, 13], [13, 14], [14, 16], [21, 22], [22, 23], [16, 21], [19, 21], [23, 23]]

  SavileRowTotalTime = 0.902
  SolverTotalTime = 0.112798
  SolverSolveTime = 0.084107
  SolverNodes = 2335
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

- *7.param*:
  total_time = 34
  op_schedule = [[1, 1], [1, 2], [2, 3], [1, 2], [3, 5], [5, 9], [9, 15], [9, 11], [15, 18], [15, 16], [5, 7], [11, 17], [18, 25], [25, 32], [32, 34], [25, 29], [32, 33], [34, 34]]

  SavileRowTotalTime = 0.999
  SolverTotalTime = 0.184289
  SolverSolveTime = 0.157068
  SolverNodes = 2967
  SolverSatisfiable = 1
  SolverSolutionsFound = 1

- *8.param*:
  total_time = 37
  op_schedule = [[1, 1], [1, 2], [1, 7], [1, 5], [7, 12], [12, 17], [23, 28], [17, 19], [17, 23], [29, 31], [28, 29], [29, 30], [31, 35], [23, 28], [35, 36], [30, 35], [36, 37], [37, 37]]

  SavileRowTotalTime = 1.029
  SolverTotalTime = 0.240688
  SolverSolveTime = 0.204155
  SolverNodes = 5919
  SolverSatisfiable = 1
  SolverSolutionsFound = 5

- *9.param*:
  total_time = 27
  op_schedule = [[1, 1], [1, 2], [1, 7], [2, 5], [5, 8], [5, 6], [19, 21], [21, 22], [6, 11], [7, 8], [21, 23], [11, 16], [24, 25], [16, 19], [19, 24], [23, 27], [25, 27], [27, 27]]

  SavileRowTotalTime = 1.25
  SolverTotalTime = 5.91529
  SolverSolveTime = 5.88497
  SolverNodes = 137429
  SolverSatisfiable = 1
  SolverSolutionsFound = 2

- *10.param*:
  total_time = 22
  op_schedule = [[1, 1], [1, 2], [1, 8], [1, 2], [16, 19], [8, 16], [8, 12], [16, 17], [16, 19], [21, 22], [17, 21], [22, 22]]

  SavileRowTotalTime = 0.729
  SolverTotalTime = 0.014168

SolverSolveTime = 0.004618
SolverNodes = 1285
SolverSatisfiable = 1
SolverSolutionsFound = 4

Large

- *1.param*:
  total_time = 45
  op_schedule = [[1, 1], [10, 18], [1, 5], [1, 7], [14, 17], [33, 41], [5, 10], [5, 14], [18, 20], [7, 14], [21, 30], [16, 18], [5, 11], [20, 23], [24, 33], [14, 24], [25, 31], [11, 16], [18, 21], [30, 37], [31, 33], [31, 38], [38, 40], [40, 43], [40, 43], [33, 40], [17, 25], [37, 40], [23, 30], [43, 45], [40, 42], [45, 45]]

  SavileRowTotalTime = 2.967
  SolverTotalTime = 30.8286
  SolverSolveTime = 30.515
  SolverNodes = 321959
  SolverSatisfiable = 1
  SolverSolutionsFound = 10

- *2.param*:
  total_time = 46
  op_schedule = [[1, 1], [1, 5], [1, 4], [1, 11], [5, 15], [28, 33], [5, 6], [6, 13], [15, 19], [18, 23], [11, 18], [18, 28], [19, 21], [19, 23], [23, 28], [23, 28], [28, 32], [11, 14], [29, 36], [5, 11], [28, 30], [14, 19], [21, 29], [32, 35], [30, 33], [33, 35], [35, 45], [36, 39], [40, 45], [35, 40], [45, 46], [46, 46]]

  SavileRowTotalTime = 2.886
  SolverTotalTime = 1.02191
  SolverSolveTime = 0.764153
  SolverNodes = 31139
  SolverSatisfiable = 1
  SolverSolutionsFound = 8

- *4.param*:
  total_time = 48
  op_schedule = [[1, 1], [1, 2], [1, 2], [1, 2], [2, 9], [2, 8], [2, 6], [9, 14], [6, 14], [9, 16], [14, 22], [6, 7], [7, 9], [14, 17], [9, 19], [29, 39], [22, 24], [16, 26], [17, 18], [2, 3], [26, 33], [37, 46], [24, 33], [24, 28], [39, 43], [43, 44], [28, 29], [29, 37], [46, 47], [46, 48], [37, 44], [48, 48]]

  SavileRowTotalTime = 2.75
  SolverTotalTime = 0.470291

SolverSolveTime = 0.233553
SolverNodes = 11297
SolverSatisfiable = 1
SolverSolutionsFound = 3

- *5.param*:
  total_time = 63
  op_schedule = [[1, 1], [1, 4], [11, 21], [1, 11], [11, 18], [4, 10], [10, 19], [18, 25], [25, 34], [25, 29], [34, 38], [35, 44], [21, 28], [4, 13], [28, 33], [38, 45], [38, 39], [29, 36], [13, 15], [44, 53], [35, 37], [36, 43], [43, 46], [25, 35], [46, 47], [39, 43], [18, 22], [47, 48], [53, 63], [54, 63], [46, 54], [63, 63]]

  SavileRowTotalTime = 3.328
  SolverTotalTime = 0.974415
  SolverSolveTime = 0.605428
  SolverNodes = 24027
  SolverSatisfiable = 1
  SolverSolutionsFound = 5

- *7.param*:
  total_time = 49
  op_schedule = [[1, 1], [1, 11], [11, 12], [1, 10], [11, 14], [10, 11], [17, 24], [11, 12], [26, 30], [12, 22], [18, 24], [12, 14], [14, 17], [24, 25], [14, 17], [25, 26], [17, 20], [29, 39], [26, 27], [39, 42], [22, 26], [24, 26], [14, 18], [30, 32], [27, 31], [42, 48], [20, 29], [27, 29], [48, 49], [31, 32], [39, 48], [49, 49]]

  SavileRowTotalTime = 2.669
  SolverTotalTime = 0.610976
  SolverSolveTime = 0.371233
  SolverNodes = 19064
  SolverSatisfiable = 1
  SolverSolutionsFound = 6

- *8.param*:
  total_time = 61
  op_schedule = [[1, 1], [1, 3], [1, 4], [1, 7], [3, 11], [4, 7], [7, 9], [19, 21], [21, 22], [11, 19], [7, 11], [19, 22], [9, 11], [11, 18], [32, 38], [22, 30], [30, 35], [4, 10], [38, 46], [35, 40], [30, 32], [30, 31], [31, 34], [10, 16], [40, 45], [35, 45], [45, 53], [45, 50], [46, 51], [53, 61], [50, 56], [61, 61]]

  SavileRowTotalTime = 2.824
  SolverTotalTime = 0.60017
  SolverSolveTime = 0.323363
  SolverNodes = 15899

- SolverSatisfiable = 1
  SolverSolutionsFound = 4

- *9.param*:
  total_time = 54
  op_schedule = [[1, 1], [1, 9], [1, 4], [1, 7], [4, 13], [13, 21], [15, 21], [7, 11], [21, 26], [4, 8], [9, 14], [14, 21], [21, 22], [11, 21], [22, 23], [21, 26], [11, 15], [26, 36], [23, 26], [21, 26], [36, 46], [36, 37], [26, 27], [27, 30], [26, 36], [36, 37], [37, 38], [37, 44], [46, 54], [36, 45], [44, 46], [54, 54]]

  SavileRowTotalTime = 2.931
  SolverTotalTime = 0.739216
  SolverSolveTime = 0.438243
  SolverNodes = 20554
  SolverSatisfiable = 1
  SolverSolutionsFound = 5

- *10.param*:
  total_time = 50
  op_schedule = [[1, 1], [1, 6], [1, 4], [1, 8], [8, 11], [16, 20], [6, 12], [12, 19], [19, 21], [6, 16], [16, 22], [26, 31], [21, 26], [31, 39], [16, 23], [33, 40], [11, 14], [23, 29], [26, 33], [4, 12], [22, 31], [29, 39], [41, 48], [20, 26], [31, 36], [40, 41], [40, 41], [41, 46], [48, 49], [46, 50], [39, 41], [50, 50]]

  SavileRowTotalTime = 2.947
  SolverTotalTime = 0.936942
  SolverSolveTime = 0.657176
  SolverNodes = 17636
  SolverSatisfiable = 1
  SolverSolutionsFound = 4

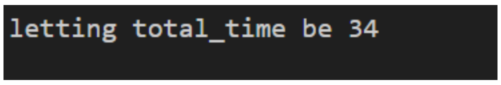
<u>Very Large</u>

- *1.param*:
  total_time = 86
  op_schedule = [[1, 1], [1, 10], [1, 2], [1, 2], [2, 8], [2, 11], [8, 18], [10, 19], [10, 14], [14, 21], [11, 12], [19, 25], [2, 6], [19, 28], [14, 23], [23, 26], [25, 32], [26, 29], [6, 10], [28, 32], [29, 34], [23, 33], [32, 40], [12, 13], [32, 35], [34, 42], [35, 43], [40, 48], [48, 53], [42, 45], [45, 52], [53, 54], [43, 51], [33, 43], [53, 55], [54, 57], [55, 62], [57, 58], [67, 70], [21, 23], [59, 64], [64, 67], [58, 68], [43, 51], [51, 56], [62, 70], [51, 59], [56, 58], [58, 66], [74, 80], [70, 71], [66, 68], [68, 74], [70, 76], [71, 77], [68, 77], [76, 78], [80, 81], [78, 86], [77, 86], [81, 82], [86, 86]]

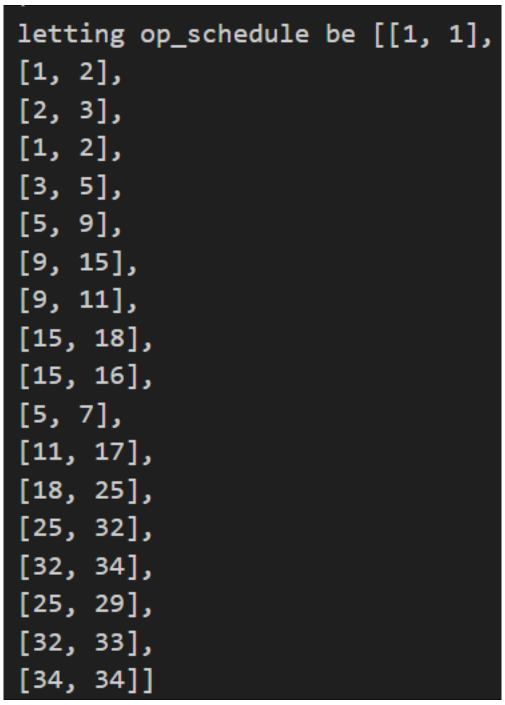
  SavileRowTotalTime = 10.245

SolverTotalTime = 17.8964
SolverSolveTime = 14.6006
SolverNodes = 201261
SolverSatisfiable = 1
SolverSolutionsFound = 13

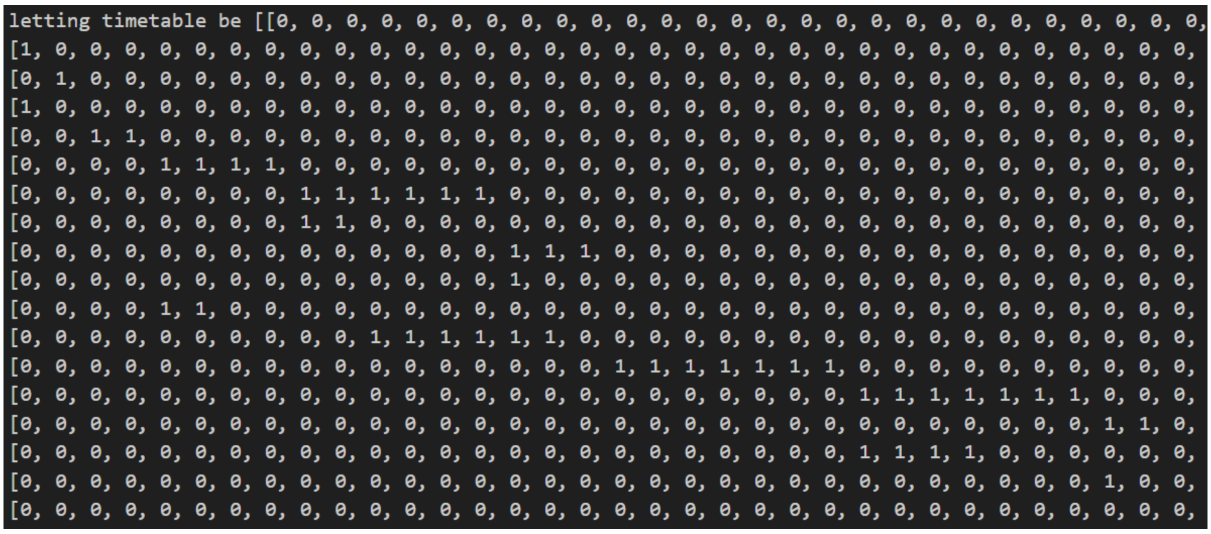
To demonstrate the correctness of my model, I will dissect one of the solutions and check whether all the constraints are being applied and that the final output is correct. I will use the solution produced using one of the medium instances: *7.param*. The *.solution* file produces has the following decision variables:



```
letting total_time be 34
```



```
letting op_schedule be [[1, 1],
[1, 2],
[2, 3],
[1, 2],
[3, 5],
[5, 9],
[9, 15],
[9, 11],
[15, 18],
[15, 16],
[5, 7],
[11, 17],
[18, 25],
[25, 32],
[32, 34],
[25, 29],
[32, 33],
[34, 34]]
```



```
letting timetable be [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

A simplified few of the output solution:

	Start	End
Operation 1	1	1
Operation 2	1	2
Operation 3	2	3
Operation 4	1	2
Operation 5	3	5
Operation 6	5	9
Operation 7	9	15
Operation 8	9	11
Operation 9	15	18
Operation 10	15	16
Operation 11	5	7
Operation 12	11	17
Operation 13	18	25
Operation 14	25	32
Operation 15	32	34
Operation 16	25	29
Operation 17	32	33
Operation 18	34	34

- The end time of the last operation is 34 which matches the value stored in *total_time*.
- All the columns in *timetable* that come before *total_time* have 1s. This means that there is always an ongoing operation.
- The start and end time follows the given duration for each operation.
- The sum of 1s for each column is less than or equal to *n_surgeon*, so the available number of surgeons isn't exceeded at each time.

- For each time slot, when there are multiple operations happening simultaneously, the sum of the required number of specialists for each of these operations doesn't exceed the available number of specialists.
- The specified order is being followed based on the assigned start and end times for each operation:
    - Operation 2 is done before operation 5.
    - Operation 3 is done before Operations 8, 5 and 13.
    - Operation 4 is done before Operations 9, 10 and 12.
    - Operation 5 is done before Operations 6 and 11.
    - Operation 6 is done before Operations 7 and 8.
    - Operation 7 is done before Operations 10, 14 and 16.
    - Operation 8 is done before Operations 10 and 16.
    - Operation 9 is done before Operations 13, 15 and 17.
    - Operation 10 is done before Operation 17.
    - Operation 11 is done before Operations 12 and 17.
    - Operation 12 is done before Operation 14.
    - Operation 13 is done before Operation 16.
    - Operation 14 is done before Operation 15.
    - Operation 15 is done before Operation 18.
    - Operation 16 is done before Operation 18.
    - Operation 17 is done before Operation 18.