Algorithms for Intelligent Decision Making (CS4210-A) TU Delft – Spring 2022 Report for Assignment 1 Group number 6

Konstantinos KRACHTOPOULOS Aditya SHANKAR

21st April 2022

Part 1

The Rotating Workforce Scheduling Problem

The present report contains experiments with two different viewpoints. For the first viewpoint, the experiments were run under Windows 10 (64 bit) on an Intel(R) Core(TM) i7-6500U 2.50 GHz, with 4 processors of 2 cores each, with 8 GB RAM and an 4 MB L3 cache. For the second viewpoint, the experiments were run under Windows 11 (64 bit) on an Intel(r) Core(TM) i7-10750H CPU 2.60 GHz, 6 cores, with 32 GB RAM and a 12 MB L3 cache.

A First Model Design and Evaluation

Listing 1: The first MiniZinc model for the Rotating Workforce Scheduling Problem

```
include "globals.mzn";

% MODEL PARAMETERS
% (N) The number of employees - groups of employees
int: groups;
% (M) Number of shifts (D, A, N)
int: numShifts;
% (W) Days in the week
int: w = 7;
% (NW) Number of days to be scheduled
int: pr = w*groups;

% Set with all shifts
set of int: S = 1..numShifts;
% Set with all shifts plus the days off
set of int: Sp = 0..numShifts;
```

```
% Set with all number of groups (or schedule wrrks)
set of int: N = 1..groups;
% Set with all days indexes in a week
set of int: W = 1..w;
% Set with all indices of the schedule
set of int: NW = 1..pr;

% (R) Employees demand per shift i per day j
array[S,W] of int: demand;
% (lw) Minimum consecutive occurances of shift s
array[S] of int: minShift;
% (uw) Maximum consecutive occurances of shift s
array[S] of int: maxShift;
% (lo) Minimum consecutive days off
int: minOff;
% (uo) Maximum consecutive days off
int: maxOff;
% (ls) Minimum consecutive shifts duration with no day off
int: minOn;
% (us) Maximum consecutive shifts duration with no day off
int: maxOn;
% (Fs) Forbidden shifts after shift s
array[S] of set of int: forbidden;
 % (F) Forbidden sequences of shifts
array[int,int] of int: forbidden3;
% Number of forbidden sequences
int: forbidden3l = length(forbidden3) div 3;

% DECISION VARIABLES
% The schedule of the first employee
array[NW] of var Sp: T;

% OBJECTIVE FUNCTION
% Search by selecting the variables with the smallest minimum value
% in their domain first.
% Then, using the 'indomain_min' value selection strategy,
% we branch on this smallest values.
solve :: int_search(T,
                    smallest,
                    indomain_min,
                    complete)
         satisfy;

% FUNCTION DECLARATIONS
% Function that performs calculations with modulus without using the
% explicit "mod" function.
function int: calc_mod(int: inp, int: divisor) =
  inp - bool2int(inp>divisor)*divisor + bool2int(inp<1)*divisor;

```

```
% CONSTRAINTS
% Constraints the demand for each day for each shift
constraint
forall(d in W, s in S)(
  count(i in 0..groups-1)(T[d+i*w] == s) =
    demand[s,d + o - bool2int((d+o)>w)*w]
);

% Constraints the number of day off assignments
constraint implied_constraint(
forall(d in W) (
  sum([bool2int(T[d+i*w] == 0) | i in 0..groups-1]) =
    groups - sum([demand[s,d + o -
                   bool2int((d+o)>w)*w] | s in S])
));

% Constraints the first day of the schedule
constraint symmetry_breaking_constraint(T[1] != 0);
% Constraints the last day of the schedule
constraint symmetry_breaking_constraint(T[pr] == 0);

% Constraints the minimum consecutive off days, and shifts
% For off days
constraint forall(i in NW)(
  (T[i] == 0) /\ (T[calc_mod(i-1, pr)] != 0) ->
      forall(j in 1..minOff-1)(T[calc_mod(i+j,pr)] == 0));
% For shifts
constraint forall(i in NW,s in S)(
    (T[i] == s) /\ (T[calc_mod(i-1,pr)] != s) ->
      forall(j in 1..minShift[s]-1)(T[calc_mod(i+j,pr)] == s)
    );

% Constraints the maximum consecutive off days, and shifts
% For off days
constraint forall(i in NW)(
  (T[i] == 0 /\ T[calc_mod(i-1,pr)] != 0) ->
    (i + maxOff > pr) \/
    exists(j in minOff..maxOff)(T[calc_mod(i+j, pr)] != 0)
);
% For shifts
constraint forall(i in NW,s in S)(
  (T[i] == s /\ T[calc_mod(i-1,pr)] != s) ->
    (i + maxShift[s] > pr) \/
    (exists(j in minShift[s]..maxShift[s])
      (T[calc_mod(i+j,pr)] != s))
);


% Constraints the minimum consecutive days without a day-off
```

```
constraint forall(i in NW)(
  (T[i] != 0 /\ T[calc_mod(i-1,pr)] == 0) ->
    forall(j in 1..minOn-1)(T[calc_mod(i+j,pr)] != 0));

% Constraints the maximum consecutive days without a day-off
constraint forall(i in NW)(
  (T[i] != 0 /\ T[calc_mod(i-1,pr)] == 0) ->
    (i + maxOn > pr) \/
    exists(j in minOn..maxOn)(T[calc_mod(i+j, pr)] == 0)
);

% Applies the forbidden shifts constraints
constraint forall(i in NW,s in S)(
  (T[i] != s) \/
  (T[calc_mod(i+1,pr)] in forbidden[s] == false)
);

% Applies the forbidden sequences constraint
constraint
forall(i in NW)(
 (forbidden3l > 0) ->
     forall(j in 1..forbidden3l)(
         forbidden3[j,..] !=
           [T[i],T[calc_mod(i+1,pr)],T[calc_mod(i+2,pr)]]
     )
);

% Constraints the offset
int: o_helper =
  min([w+1] ++
  [d-1 | d in 2..w where sum(demand[..,d]) > sum(demand[..,d-1])]
  );
% Calculates the offset
int: o = if o_helper <= w then o_helper else 0 endif;

% Checks for infeasibile weekly fluctuation
constraint
  forall(s in S, i in W, j in maxShift[s]+1..2*minShift[s]-1,
    k in j-minShift[s]..minShift[s]-1) (
    demand[s,calc_mod(i+k, w)] >=
    demand[s,i] - demand[s,calc_mod(i-1, w)] +
    demand[s,calc_mod(i+j-1, w)] -
    demand[s, calc_mod(i+j, w)]
  );

% OUTPUT
array[1..4] of string: shifts_str= ["-", "1", "2", "3"];
output ["["] ++ [shifts_str[fix(T[calc_mod(i-o,pr)])+1] ++
  if i == pr then "]\n"
  else ","
  endif | i in NW];
```

A.1 Description

The first algorithm designed for the Rotating Workforce Scheduling Problem exploits the fact that the generated schedule is cyclic and hence only calculates the schedule for the first employee (or employees group). As a result, the variables is an array T, of length $n \times w$, where n is the number of employees, and w is the days in a week. The constraints formation was based on [1].

Redundant Decision Variables and Channelling Constraints. Since this model consists of one viewpoint, there are no channelling constraints connecting the different decision variables. As stated above, the only decision variable of this model is array T, denoting the schedule of the first employee for the next n weeks. Moreover, parameter o is defined, denoting the offset of the first element of T from the first day of the week. In order to calculate this offset we use a helper parameter o_helper, as the minimum day index in which the total demand for the next day is lower than the demand of the current day.

```
int: o_helper =
  min([w+1] ++
  [d-1 | d in 2..w where sum(demand[..,d]) > sum(demand[..,d-1])]
  );
% Calculates the offset
```

If there exists such an index, then the offset is set equal to this value; otherwise, it is set equal to zero (i.e. the first value of schedule T is Monday).

```
int: o = if o_helper <= w then o_helper else 0 endif;
```

Implied Constraints. Since a constraint to specify the number of people that are needed per day per shift already exists, the number of people having a day off is explicitly stated. However, a constraint to specify the latter is added, because this constraint limits the domains of the decision variables. The predicate `implied_constraint` is used as a good practice.

```
constraint implied_constraint(
forall(d in W) (
  sum([bool2int(T[d+i*w] == 0) | i in 0..groups-1]) =
    groups - sum([demand[s,d + o -
                  bool2int((d+o)>w)*w] | s in S])
));
```

Moreover, after some thought, it becomes clear that when the demand for shifts greatly fluctuates within a week (f.i. much lower demand during the weekends). Hence, it may not be possible to satisfy the minimum and maximum consecutive shifts with no day off constraints. As a result, a constraint that detects possible infeasibility due to large demand fluctuations is introduced.

```
constraint
  forall(s in S, i in W, j in maxShift[s]+1..2*minShift[s]-1,
    k in j-minShift[s]..minShift[s]-1) (
    demand[s,calc_mod(i+k, w)] >=
    demand[s,i] - demand[s,calc_mod(i-1, w)] +
    demand[s,calc_mod(i+j-1, w)] -
157     demand[s, calc_mod(i+j, w)]
158   );
```

It is worth noting that since the constraint plays a major role in detecting infeasibility, it was deemed that it should be modelled as a normal constraint.

Symmetry-Breaking Constraints. Because any day can in principal be the first day of the schedule in a Rotating Workforce Schedulling problem, symmetry breaking is needed. We can reduce the number of solutions by stating that the first day of the schedule must be a working day. We limit the solution space even further by stating that the last day of the schedule should be a working day. Of course, the above statements require that there is at least one working day and one non-working day.

```
83 constraint symmetry_breaking_constraint(T[1] != 0);
```

```
85 constraint symmetry_breaking_constraint(T[pr] == 0);
```

These constraints are modelled using the predicate `symmetry_breaking_constraint` predicate, following the MiniZinc good practices.

Inference Annotations. We notice that most of the constraints created are linear equalities/inequalities. After experimentation, we deemed that it is best not to add any inference annotation to them, as the backends can handle them efficiently. Moreover, bounds consistency annotation was initially suggested for the `exists` constraints(lines 103,123,130), and for the non-linear inequality constraint (line 138), as in this way we could observe a better performance. However, as we will see in the evaluation, no effect was orbserved with the addition of these annotations. Hence, the default consistency type will be selected for each backend.

Search Annotation. Unlike other problems, the Rotating Workforce Scheduling problem doesn't appear to give any intuition concerning what search strategy to follow. The aforementioned claim was validated by testing all different combinations of the available variable selection and value selection strategies in selected example instances. Namely, we noticed that there is no combination that dominates the others in all examples. Nevertheless, the combination that was proven to give the most promising results was the `indomain_min` as the variable selection strategy, and the `smallest` as the value selection strategy. With this combination, the variable with the smallest value inside its domain is selected for the next branch (e.g. variable `d` and `D(d)` is its domain). For the branching, the left branch is assigned the minimum value of the domain (i.e. `v = min(D(d))`, while the right branch is assigned all the other values (`v != min(D(d))`).

A.2 Implementation

The described model, with the prescribed comments, is uploaded as file `CSP_solution_satisfy.mzn`.

Compilation and Running Instructions. To compile and run `CSP_solution_satisfy.mzn` one must supply an input file for the following parameters:

1. *groups*: (`int`) Number of employees
2. *numShifts*: (`int`) Number of available shifts
3. *demand*: (`array[1..numShifts,1..7] of int`) Demand of people per shift per day

4. *minShift*: (`array[1..numShifts] of int`) Minimum consecutive working days per shift
5. *maxShift*: (`array[1..numShifts] of int`) Maximum consecutive working days per shift
6. *minOff*: (`int`) Minimum number of consecutive days off
7. *maxOff*: (`int`) Maximum number of consecutive days off
8. *minOn*: (`int`) Minimum number of consecutive working days
9. *maxOn*: (`int`) Maximum number of consecutive working days
10. *forbidden*: (`array[1..numShifts] of set of int`) Set of forbidden shifts after each shift
11. *forbidden3*: (`array[int,1..3] of int`) Forbidden shift sequences of length 3.

Moreover, for an enchanced performance, especially when using Chuffed solver, the user should make use of the `-f` option, which corresponds to free search. Assuming the `mzn-backend` is one of the available backends, the model can be compiled and run for file `input_file.dzn` by executing the following command at the command line: `minizinc --solver mzn-backend CSP_solution_satisfy.mzn input_file.dzn -f`.

Sample Test-Run Command.

```
> minizinc --solver Chuffed CSP_solution_satisfy.mzn "Example1.dzn" -f
[...snip...]
----------
Schedule: [3,-,-,1,1,1,1,-,-,-,-,1,1,1,1,-,-,2,2,2,3,
3,3,3,-,-,2,2,2,2,2,2,-,-,2,2,3,3,3,3,-,-,1,1,1,1,2,2,
-,-,1,1,3,3,3,-,-,2,2,2,2,3,3]
----------
==========
Finished in 959msec
```

B Second Model Design and Evaluation

Listing 2: The second MiniZinc model for the Rotating Workforce Scheduling Problem

```
include "globals.mzn";

% Set the parameters
int: groups; % The number of employees - groups of employees
int: numShifts; % Number of shifts (D, A, N)
int: w = 7; % Days in the week
set of int: S = 1..numShifts;
set of int: Sp = 0..numShifts;
set of int: N = 1..groups;
set of int: W = 1..w;
```

```
array[S,W] of int: demand; % number of employees needed to be
  assigned per shift i per day j
array[S] of int: minShift;
array[S] of int: maxShift;
int: minOff;
int: maxOff;
int: minOn;
int: maxOn;
int: pr = groups*w;
array[S] of set of int: forbidden;
array[int,int] of int: forbidden3;
int: forbidden3l = length(forbidden3) div 3;



%decision variables in new perspective
array[1..groups,1..w,0..numShifts] of var 0..1: T_new;
array[1..groups*w] of var Sp: T;
var 0..groups: free;


solve satisfy;

%function to calculate modulus
function int: calc_mod(int: inp, int: divisor) =
  inp - bool2int(inp>divisor)*divisor + bool2int(inp<1)*divisor;

%to have exactly one shift selected
constraint
forall(worker in N,day in W)(

    count_eq(T_new[worker,day,..],1,1)
);

%To convert to desired output format
constraint
forall(i in 1..groups, j in 1..w, k in 0..numShifts)(
  (T_new[i,j,k] == 1) ->(T[(i-1)*w + j] = k)
);

%demand constraints
constraint
forall(s in S,d in W)(
  (sum(x in N)(T_new[x,d,s])) == demand[s,((d+o-1) mod w)+1]
);

constraint implied_constraint(
forall(d in W)(
  (sum(x in N)(T_new[x,d,0])) == groups - (sum(s in
    S)(demand[s,((d+o-1) mod w)+1]))
));

%symmetry breaking constraints
constraint symmetry_breaking_constraint(T_new[1,1,0] != 1);
constraint symmetry_breaking_constraint(T_new[groups,w,0]== 1);

%shift constraints
%minShift maxShift
constraint
forall(s in S, n in 1..groups, d in 1..w)(
 if(d > 1) then(
    if(T_new[n,d,s] == 1 /\ T_new[n,d-1,s]==0) then(
      forall(x in 0..minShift[s]-1)(
        T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
          (((n-1)*w + d-1 + x) mod w) + 1, s] == 1
      )

      /\
      sum(x in minShift[s]..maxShift[s])(T_new[((((n-1)*w + d-1 +
       x) div w) mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1,
       s]) < (maxShift[s] - minShift[s] + 1)
    )
    else true endif
  )

  elseif(d == 1 /\ n > 1) then(
    if(T_new[n,d,s] == 1 /\ T_new[n-1,w,s]==0) then(
        forall(x in 0..minShift[s]-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, s] == 1
        )

        /\
        sum(x in minShift[s]..maxShift[s])(T_new[((((n-1)*w + d-1 +
         x) div w) mod groups) + 1, (((n-1)*w + d-1 + x) mod w) +
         1, s]) < maxShift[s] - minShift[s] + 1
    )
    else true endif
  )
  elseif(d == 1 /\ n == 1) then(
    if(T_new[n,d,s] == 1 /\ T_new[groups,w,s]==0) then(
        forall(x in 0..minShift[s]-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, s] == 1
        )

        /\
        sum(x in minShift[s]..maxShift[s])(T_new[((((n-1)*w + d-1 +
          x) div w) mod groups) + 1, (((n-1)*w + d-1 + x) mod w) +
          1, s]) < maxShift[s] - minShift[s] + 1
        )
        else true endif
    )
    else true
    endif
);

%minOff and maxOff
constraint
forall(n in 1..groups, d in 1..w)(
 if(d > 1) then(
    if(T_new[n,d,0] == 1 /\ T_new[n,d-1,0]==0) then(
      forall(x in 0..minOff-1)(
        T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
          (((n-1)*w + d-1 + x) mod w) + 1, 0] == 1
      )

      /\
      sum(x in minOff..maxOff)(T_new[((((n-1)*w + d-1 + x) div w)
        mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) <
        maxOff - minOff + 1
    )
    else true endif
  )

  elseif(d == 1 /\ n > 1) then(
    if(T_new[n,d,0] == 1 /\ T_new[n-1,w,0]==0) then(
        forall(x in 0..minOff-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, 0] == 1
        )

        /\
        sum(x in minOff..maxOff)(T_new[((((n-1)*w + d-1 + x) div w)
          mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) <
          maxOff - minOff + 1
    )
    else true endif
  )
  elseif(d == 1 /\ n == 1) then(
    if(T_new[n,d,0] == 1 /\ T_new[groups,w,0]==0) then(
        forall(x in 0..minOff-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, 0] == 1
        )

        /\
        sum(x in minOff..maxOff)(T_new[((((n-1)*w + d-1 + x) div w)
          mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) <
          maxOff - minOff + 1
      )
      else true endif
  )
  else true
  endif
);

%minOn and maxOn
constraint
forall(n in 1..groups, d in 1..w)(
 if(d > 1) then(
    if(T_new[n,d,0] == 0 /\ T_new[n,d-1,0]==1) then(
      forall(x in 0..minOn-1)(
        T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
          (((n-1)*w + d-1 + x) mod w) + 1, 0] == 0
      )

      /\
      sum(x in minOn..maxOn)(T_new[((((n-1)*w + d-1 + x) div w) mod
        groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) > 0
    )
    else true endif
  )

  elseif(d == 1 /\ n > 1) then(
    if(T_new[n,d,0] == 0 /\ T_new[n-1,w,0]==1) then(
        forall(x in 0..minOn-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, 0] == 0
        )

        /\
        sum(x in minOn..maxOn)(T_new[((((n-1)*w + d-1 + x) div w)
          mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) > 0
    )
    else true endif
  )
  elseif(d == 1 /\ n == 1) then(
    if(T_new[n,d,0] == 0 /\ T_new[groups,w,0]==1) then(
        forall(x in 0..minOn-1)(
          T_new[((((n-1)*w + d-1 + x) div w) mod groups) + 1,
            (((n-1)*w + d-1 + x) mod w) + 1, 0] == 0
        )

        /\
        sum(x in minOn..maxOn)(T_new[((((n-1)*w + d-1 + x) div w)
          mod groups) + 1, (((n-1)*w + d-1 + x) mod w) + 1, 0]) > 0
      )
      else true endif
  )
  else true
  endif
);

%forbidden sequences
%length 2

constraint
forall(n in 1..groups, d in 1..w, s in S)(
  if(T_new[n,d,s] == 1 /\ length(forbidden[s]) > 0) then(
    forall(x in forbidden[s], i in 1..length(forbidden[s]))(
      T_new[((((n-1)*w + d-1 + i) div w) mod groups) + 1, (((n-1)*w
       + d-1 + i) mod w) + 1, x] != 1
    )
  )
  else true endif
);

%length 3

constraint
forall(n in 1..groups, d in 1..w, s in S)(
  if(T_new[n,d,s] == 1 /\ forbidden3l > 0) then(
    forall(x in 1..forbidden3l)(
      (T_new[((((n-1)*w + d-1 + 1-1) div w) mod groups) + 1,
       (((n-1)*w + d-1 + 1-1) mod w) + 1, forbidden3[x,1]] != 1) \/
      (T_new[((((n-1)*w + d-1 + 2-1) div w) mod groups) + 1,
       (((n-1)*w + d-1 + 2-1) mod w) + 1, forbidden3[x,2]] != 1) \/
      (T_new[((((n-1)*w + d-1 + 3-1) div w) mod groups) + 1,
       (((n-1)*w + d-1 + 3-1) mod w) + 1, forbidden3[x,3]] != 1)
    )
  )
  else true endif
);

%offset breaking

% Constraints the offset
int: o_helper =
  min([w+1] ++
  [d-1 | d in 2..w where sum(demand[..,d]) > sum(demand[..,d-1])]
  );
% Calculates the offset
int: o = if o_helper <= w then o_helper else 0 endif;

```

```
array[1..4] of string: shifts_str= ["-", "1", "2", "3"];
output ["["] ++ [shifts_str[fix(T[calc_mod(i+pr - o,pr)])+1] ++
  if i == pr then "]\n"
  else ","
  endif | i in 1..pr];
output ["Offset: \(o)\n"];
```

B.1 Description

Unlike the first model, the second one makes use of a 3D binary array of dimensions (n, w, s), where n is the number of employees (groups), w is the number of days in a week, and s is the number of shifts, i.e., Day, Afternoon, Night, and Holiday. For each (employee, day), the corresponding shift is assigned the value 1 and all others are set to zero. The shift indices are 0 for holidays, 1 for day shift, 2 for afternoon shift, and 3 for night shift.

Redundant Decision Variables and Channelling Constraints. This model, has channelling constraints linking the different output variables: T and T_new. T_new is the 3D binary array, and T is the final output array that translates the results of the binary table to a format similar to the first model, i.e., a 1D array. This has been done using the following constraint:

```
%To convert to desired output format
constraint
forall(i in 1..groups, j in 1..w, k in 0..numShifts)(
  (T_new[i,j,k] == 1) ->(T[(i-1)*w + j] = k)
);
```

Implied Constraints. Similar to the first model, we make use of implied_constraint for the number of days off.

```
constraint implied_constraint(
forall(d in W)(
  (sum(x in N)(T_new[x,d,0])) == groups - (sum(s in
    S)(demand[s,((d+o-1) mod w)+1]))
));
```

Symmetry-Breaking Constraints. The following constraints break symmetry. The first and last days are forced to be a working and non-working day respectively.

```
constraint symmetry_breaking_constraint(T_new[1,1,0] != 1);
constraint symmetry_breaking_constraint(T_new[groups,w,0]== 1);
```

Inference Annotations. No inference annotations were used for this version too, for the same reasons mentioned in the earlier model.

Search Annotation. Unlike the first model, no significant performance gain was seen when using the indomain_min or the largest value selection strategy. The intuition behind this was that 1's occur less frequently in the binary table compared to 0's, so initially assigning more variables to 1 would result in conflicts being detected earlier, but the performance in fact degraded as shown in the experimental results. However, running the models with the free search parameter (-f) greatly improves performance.

B.2 Implementation

The described model, with the prescribed comments, is uploaded as file: `CSP_solution_version_2_satisfy.mzn`.

Compilation and Running Instructions. To compile and run: `CSP_solution_version_2_satisfy.mzn` one must supply an input file for the following parameters:

1. *groups*: (`int`) Number of employees
2. *numShifts*: (`int`) Number of available shifts
3. *demand*: (`array[1..numShifts,1..7] of int`) Demand of people per shift per day
4. *minShift*: (`array[1..numShifts] of int`) Minimum consecutive working days per shift
5. *maxShift*: (`array[1..numShifts] of int`) Maximum consecutive working days per shift
6. *minOff*: (`int`) Minimum number of consecutive days off
7. *maxOff*: (`int`) Maximum number of consecutive days off
8. *minOn*: (`int`) Minimum number of consecutive working days
9. *maxOn*: (`int`) Maximum number of consecutive working days
10. *forbidden*: (`array[1..numShifts] of set of int`) Set of forbidden shifts after each shift
11. *forbidden3*: (`array[int,1..3] of int`) Forbidden shift sequences of length 3.

Moreover, for an enchanced performance, especially when using Chuffed solver, the user should make use of the `-f` option, which corresponds to free search. Assuming the `mzn-backend` is one of the available backends, the model can be compiled and run for file `input_file.dzn` by executing the following command at the command line: `minizinc --solver mzn-backend CSP_solution_version_2_satisfy.mzn input_file.dzn -f`. To match the specifications given in the assignment description, the output was shifted to the right by the offset.

Sample Test-Run Command.

```
> minizinc --solver Chuffed CSP_solution_version_2_satisfy.mzn "Example1.dzn"
-f
Running CSP_solution_version_2_satisfy.mzn with Example1.dzn
[3,-,-,2,2,2,3,3,3,-,-,1,1,1,1,3,3,3,-,-,-,-,1,1,1,2,2,2,2,-,-,-,-,1,1,1,1,1,1
,1,-,-,2,2,3,3,3,3,-,-,2,2,2,2,2,2,-,-,2,2,3,3,3]
----------
Finished in 581msec
```

Technology	LCG	CP
Solver	Chuffed	Gecode
Backend	mzn-chuffed	mzn-gecode
Example	time	time
1	0.59	0.58
2	0.65	0.65
3	0.86	15.58
4	0.75	0.67
5	1.08	0.65
6	0.98	0.55
7	2.56	0.86
8	1.66	296.02
9	1.62	t/o
10	0.88	503.78
11	1.43	t/o
12	0.79	t/o
13	0.81	t/o
14	0.68	7.67
15	47.01	t/o
16	0.88	t/o
17	0.85	t/o
18	1.66	26.24
19	9.73	t/o
20	5.2	t/o

Table 1: Results for our first Rotating Workforce Scheduling model. In the 'time' column, if the reported time is less than the time-out (600 seconds here), then a feasible solution has been found for the Example referenced in the first column; else the time-out 't/o' indicates that no feasible solution was found before timing out.

C Experiments with Satisfiability

C.1 First model

Experiments. Table 1 gives the results for various instances of our first Rotating Workforce Scheduling model. The time-out was 600 seconds.

Analysis. We observe that the chosen LCG backend clearly surpasses the CP one, as it is able to provide a feasible solution for all instances. On the other hand, mzn-gecode backend only finds a feasible solution before time-out for 11/20 instances. The LCG backend scales much better, as it can handle the largest instances (15,19,20), unlike the CP instance, which fails to terminate for these examples. On small instances, the results of the two backends are comparable, while the CP backend is marginally faster. Because of the overall superiority of the mzn-chuffed compared to the rest evaluated, further analysis will be only based on these results. Finally, it is worth mentioning that mzn-coin-bc backend (MIP) was also evaluated, but since it performed worse than the LCG backend in all examples, its results weren't included in this report.

Redundant Decision Variables and Channelling Constraints. The model did not contain any Redundant Decision Variables of Channelling Constraints. Therefore, no evaluation is performed in this paragraph.

Implied Constraints. In order to evaluate the importance of the implied constraint (line 76), we executed the model without this constraint, for two representative instances; a small one (Example 7) and a larger one (Example 15). Both models reach the time-out without the implied constraint! This shows that it indeed plays a major role in the fast solving process of the model. On the other hand, for the implied constraint in line 151, its runtime effect is not visible in the small instance, but it provides a 53% improvement in the runtime of the large instance.

Symmetry-Breaking Constraints. The model has been built with the assumption that the first day of the schedule is a working day and the last day is a day off. If these constraints get removed, then the model becomes corrupted, and can't provide correct feasible solutions. In this essensce, the importance of the two symmetry constraints is huge, since they allow the construction of a simpler model.

Inference Annotations. As it was mentioned in the model description, no runtime performance was observed when bounds consistency annotation was added in the constraints.

Search Annotation. Concerning the search annotation, the specified annotation was `smallest` for variable selection and `indomain_min` for the value selection. Moreover, during the models execution with Chuffed, the free search parameter `-f` was added, giving the possibility to the algorithm to perform free search. We can understand the importance of both assignment by running the model without them. It is very interesting to point out that when running the model for the large dataset, time-out is reached when any of the two is missing! We were not able to find many information about Chuffed "free-search" option, but it seems that when combined with a user-specified search strategy, it performs very well. Lastly, the chosen search strategy performs better than all the other combinations of value/variable selection strategies.

C.2 Second model

Experiments. Table 2 gives the results for various instances of our second Rotating Workforce Scheduling model. The timeout was set to 3600 seconds.

Analysis. The dominance of the LCG solver becomes more apparent when testing for satisfiability with the second model. We see that with the Chuffed solver, 17/20 instances solve before the timeout of 3600 seconds. However, the CP solver gecode is only able to satisfy 6 instances within this limit compared to the 11 instances in the first model. The advantage of the first model is that a linear array is very suitable for capturing the cyclic aspect of the schedules. Modulo indexing in that case was only required along one dimension, i.e., when the indexs reached the end of the 1D array. On the other hand, the second design requires modulo indexing for both the days of the week and the number of employees, making it computationally more intensive and inefficient. Furthermore, since their are 4 binary shift variables for (D,A,N,-), the memory occupied is also 4 times larger compared to the first model.

Technology	LCG	CP
Solver	Chuffed	Gecode
Backend	mzn-chuffed	mzn-gecode
Example	time	time
1	0.41	0.71
2	0.40	0.52
3	0.55	t/o
4	0.54	35.37
5	0.50	25.99
6	0.39	0.65
7	53.74	0.80
8	0.53	t/o
9	1.41	t/o
10	0.78	t/o
11	924.24	t/o
12	1.29	t/o
13	0.77	t/o
14	0.59	t/o
15	t/o	t/o
16	1.03	t/o
17	0.93	t/o
18	6.56	t/o
19	t/o	t/o
20	t/o	t/o

Table 2: Results for our second Rotating Workforce Scheduling model. In the 'time' column, if the reported time is less than the time-out (3600 seconds here), then a feasible solution has been found for the Example referenced in the first column; else the time-out 't/o' indicates that no feasible solution was found before timing out.

Redundant Decision Variables and Channelling Constraints. The only redundant decision variables were T_new and T, and the influence of the redundant constraints relating the two were checked for performance differences. However, no clear performance gain was observed upon including them. This is expected because the the relation between the two is very straightforward as the constraints mainly seek to map the results of the binary array onto the single-dimensional array for printing out the final answer.

Implied Constraints. In order to evaluate the importance of the implied constraint, we executed the model without this constraint, on the smaller examples from 1 to 10. The table 3 shows the execution times when executing without the implied constraints. A time out of 100 seconds was used because all 10 instances were solved to satisfaction within 60 seconds when including the implied constraints, so a larger value than 100 seconds was deemed unnecessary for evaluation. Observe that in all 10 instances, the performance without the implied constraints is strictly worse. We can therefore only expect the improved performance by including the implied constraints on the larger examples (11-20) to be much more significant.

	With implied constraint	without implied constraint
Example	time	time
1	0.41	0.65
2	0.40	0.44
3	0.55	0.75
4	0.54	0.6
5	0.50	0.57
6	0.39	0.45
7	53.74	t/o
8	0.53	0.6
9	1.41	t/o
10	0.78	5.20

Table 3: Results for our second Rotating Workforce Scheduling model. In the 'time' column, if the reported time is less than the time-out (100 seconds here), then a feasible solution has been found for the Example referenced in the first column; else the time-out 't/o' indicates that no feasible solution was found before timing out.

Symmetry-Breaking Constraints. As mentioned earlier, the symmetry-breaking constraints greatly reduce the search space. Instance 9 originally took 1.41 seconds. However, removing the symmetry-breaking constraints increased its run time to 24.30 seconds. The earlier instances had a slight increase in execution times too. Given the larger size of the binary table compared to the 1D array, the inclusion of the symmetry breaking becomes more necessary. We however must mention that removing the symmetry breaking constraint does not retain the same expected output solutions. This test was only done to see the effect in terms of computational speedup and not correctness.

Search Annotation. In the binary array, T_new, since the value 1 occurs fewer times, we thought using the strategy `indomain_min` with `largest` value selection strategy would give better performance. The assumption was that forcing more variables to take on the value 1 would lead to earlier pruning in the search tree. However, the general performance was even worse upon choosing this strategy as shown in table. Again, a timeout of 100 seconds was chosen. Results are shown in table 4

D Experiments with Free Weekends Maximization

D.1 First model

Description The next task for our Rotating Workforce Scheduling models was to produce a schedule that maximizes the free weekends. That is a process with much importance in real-life. For this reason, a new array of variables `f` was introduced, denoting the free weeks. Hence, the optimization objective for the algorithm is to maximize the sum of `f`.

```
var 0..groups: f;
```

The corresponding constraint is:

```
constraint f = sum(i in 1..groups)(T[calc_mod(w*i-o-1, pr)] == 0 /\
 T[calc_mod(pr+w*i-o,pr)] == 0);
```

	Without search annotation	with search annotation
Example	time	time
1	0.41	0.46
2	0.40	0.45
3	0.55	1.38
4	0.54	0.73
5	0.50	1.67
6	0.39	0.45
7	53.74	0.83
8	0.53	t/o
9	1.41	t/o
10	0.78	t/o

Table 4: Search annotation performance in our second model In the 'time' column, if the reported time is less than the time-out (100 seconds here), then a feasible solution has been found for the Example referenced in the first column; else the time-out 't/o' indicates that no feasible solution was found before timing out.

Except from the objective function the algorithm remained unchanged. The implementation, and expected inputs are also the same.

Experiments. Table 5 gives the results for various instances of our first optimization model. The time-out was 3600 seconds.

Analysis. We observe that the mzn-chuffed backend again surpasses the CP backend, just as in the case of satisfiability. This is an expected behavior, since Chuffed solver seems to be more able to limit the solutions space, and reach to solutions faster. The solver is able to solve 12/20 instances to optimality. However, a satisfiable solution is found for 19/20 within 10 seconds. Again the backend mzn-coin-bc is worse then mnz-chuffed backend for all the instances tested. For this reason, full experimentation with this backend is omitted.

Redundant Decision Variables and Channelling Constraints. The model did not contain any Redundant Decision Variables of Channelling Constraints. Therefore, no evaluation is performed in this paragraph.

Implied Constraints. No additional implied constraints are added. At this point, it is worth noting that in the paper given together with the project description, another implied constraint was implemented using the `global_cardinality_low_up` predicate. Even though it was stated that this constrained improved the runtime, we were not able to identify any improvement. Instead, we observed a small decrease in the performance. Namely, for Example 10, which is considered representative, as it is solved by Chuffed solver in `1545` seconds, the addition of the cardinality implied constraint resulted in a runtime of `2100` seconds.

Symmetry-Breaking Constraints. No change was made in the symmetry breaking constraints compared to the standard model.

Technology	LCG		CP	
Solver	Chuffed		Gecode	
Backend	mzn-chuffed		mzn-gecode	
n	Free	time	Free	time
1	2	0.71	2	0.82
2	3	0.89	3	3.1
3	5	0.95	5	558.27
4	3	0.71	3	5.32
5	5	0.77	5	9.53
6	2	0.71	2	0.56
7	11	t/o	11	t/o
8	12	1.04	12	283.36
9	35	t/o	-	t/o
10	15	1545.59	14	t/o
11	6	t/o	t/o	t/o
12	8	6.05	t/o	t/o
13	6	3.29	t/o	t/o
14	4	0.72	4	1091.35
15	13	t/o	-	t/o
16	9	778.02	-	t/o
17	11	t/o	-	t/o
18	23	t/o	18	t/o
19	28	t/o	-	t/o
20	43	t/o	-	t/o

Table 5: Results for our first RWS optimization model. In the ‘time’ column, if the reported time is less than the time-out (3600 seconds here), then the reported objective value in the ‘free’ column was proven optimal; else the time-out is indicated by ‘t/o’ and the reported objective value is either the best value found, but not proven optimal, before timing out, or ‘–’, indicating that no feasible solution was found before timing out.

Inference Annotations. Inference annotations were again tested in the same way as in the satisfiability problem, but did not give any performance improvement. Hence, it was left to the solvers to identify the best consistency method in the constraints.

Search Annotation. All the different combinations of value/variable selection strategies were tested, and, as in the satisfiability case, the combination performing best was `smallest` for variable selection and `indomain_min` for variable selection.

Intermediate solutions - Trace plot Apart from observing how many instances were solved to optimality, it is also very interesting to review the traces of intermediate solutions found for every instance. These solutions depict feasible solutions which were the current optimum at the time of execution. Intermediate solutions serve as a good alternative when a problem can't be solved to optimality, because these values often approximate the optimal solution quite well. In the case of the first viewpoint, at least one intermediate solution was produced for every instance. In 1 we can see the trace plot for all 20 instances.

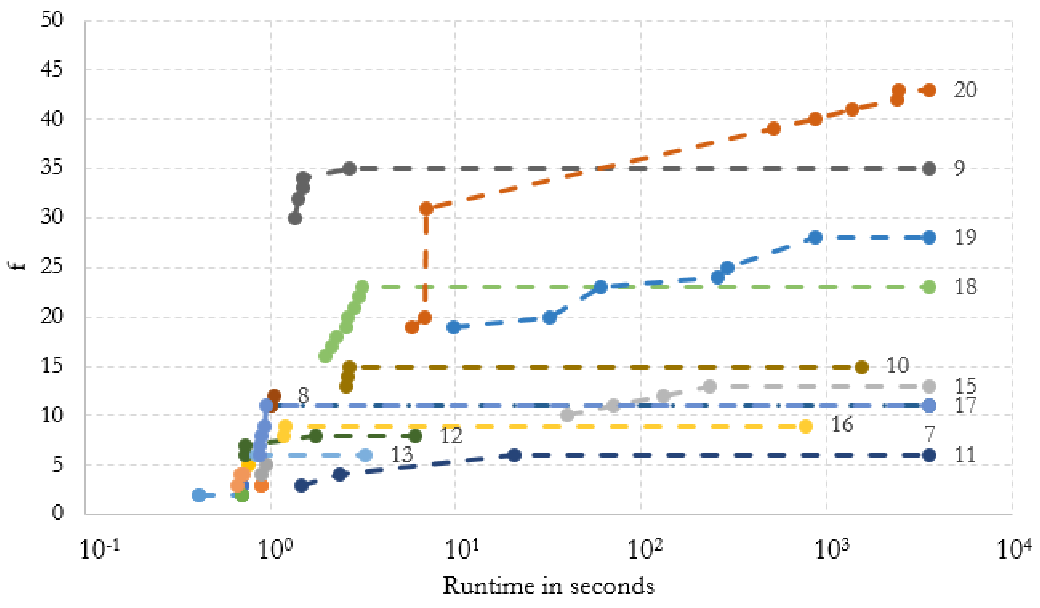

Figure 1: Trace plot of all intermediate solution reported for the 20 instances with Chuffed solver. The instance number is visible when possible.

D.2 Second model

Description When solving for satisfiability, we already observed that the CP solver, gecode times out on a majority of the instances. Hence, we do not present the results for gecode for the optimization task as it does not add any new information. Hence we feel it is sufficient to discuss the results with the Chuffed solver alone. The only difference in the code was the addition of another variable to maximize the number of free weekends. The variable and its related constraints are given below:

```
var 0..groups: free;
```

It is maximized using:

```
%maximize the number of free weekends
solve maximize free;
```

And constrained using:

```
constraint
  if(o == 6) then(
    free = sum(x in 1..groups)((T_new[((x-1+groups) mod groups) +
      1,w,0] = 1) /\ (T_new[x,w-o,0]=1))
  )
  else(
    free = sum(x in 1..groups)((T_new[x,w-o-1,0] = 1) /\
      (T_new[x,w-o,0]=1))
  )
  endif
;
```

Technology	LCG	
Solver	Chuffed	
Backend	mzn-chuffed	
n	Free	time
1	2	0.41
2	3	0.41
3	5	0.7
4	3	0.55
5	5	0.51
6	2	0.41
7	11	t/o
8	12	0.56
9	35	t/o
10	15	296.56
11	3	t/o
12	8	10.08
13	6	9.9
14	4	0.62
15	_	t/o
16	9	1472.01
17	11	t/o
18	23	t/o
19	_	t/o
20	_	t/o

Table 6: Results for our second RWS optimization model. In the 'time' column, if the reported time is less than the time-out (3600 seconds here), then the reported objective value in the 'free' column was proven optimal; else the time-out is indicated by 't/o' and the reported objective value is either the best value found, but not proven optimal, before timing out, or '–', indicating that no feasible solution was found before timing out.

Experiments. Table 1 gives the results for various instances of our second optimization model. The time-out was 3600 seconds.

Analysis. Despite being only able to find a solution for 17/20 instances, the second model matches the first model in terms of the number of optimal solutions, i.e. 12/20.

Redundant Decision Variables and Channelling Constraints. No additional redundant/channelling constraints were added apart from the ones already mentioned when checking for satisfiability.

Implied Constraints. No additional implied constraints were added apart from the ones already mentioned when checking for satisfiability.

Symmetry-Breaking Constraints. No change was made in the symmetry breaking constraints compared to the standard model.

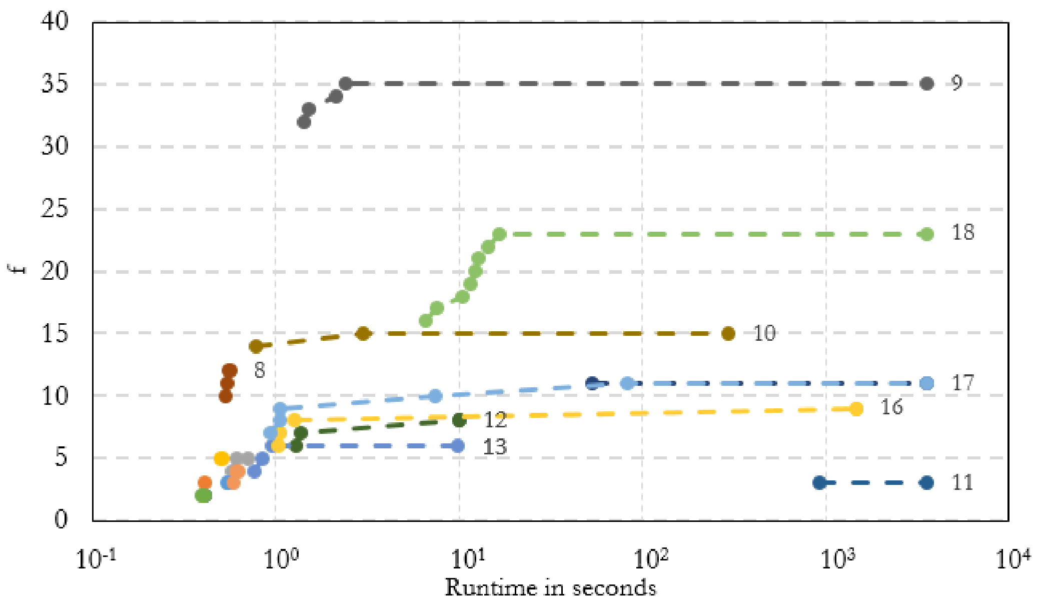

Figure 2: Trace plot of all intermediate solution reported for the 20 instances with Chuffed solver. The instance number is visible when possible.

Inference Annotations. No inference annotations used.

Search Annotation. The results from the satisfiability scenario already indicate a degradation in performance when using a customized search annotation. Hence, no other option was explored.

Intermediate solutions - Trace plot Apart from observing how many instances were solved to optimality, it is also very interesting to review the traces of intermediate solutions found for every instance. These solutions depict feasible solutions which were the current optimum at the time of execution. Intermediate solutions serve as a good alternative when a problem can't be solved to optimality, because these values often approximate the optimal solution quite well. In the case of the first viewpoint, at least one intermediate solution was produced for every instance. In 2 we can see the trace plot for all 20 instances.

E Modelling Strategies

For the first model, we treated the generated schedule as a 1-dimensional array that corresponds to the schedule of the first employee. The assumption made here was that the schedule is cyclic. In this manner, the second employee will have the schedule of the first employee, with one week of offset, the third will have two weeks of offset, etc. Of course, the generated schedule has a length of n weeks, where n is the number of employees. That is the only variable of the problem, with a domain of values from `0` to `m`, where `m` is the number of different shifts.
The implemented constraints aimed to reduce the search space of the available solutions by constraining the decision variables domains. Most constraints can be comprehended by intuition (f.i. demand, min/max consecutive on/off days). Moreover, implied and symmetry breaking constraints are introduced so that the solution space is further decreased and we reach a solution

faster. The speed of search was also influenced by the introduction of search annotations for the variable selection and the value selection during the branching. It was proven that for our case, the best search strategy was to select the variable with the minimum value in its domain (`indomain_min`) and branch on this minimum value (`smallest`). In terms of consistencies, no specific annotation was proven to be more effective than the default of each backend, and hence, it was left for the backend to handle this.

In the second model, it was difficult to include many global constraints because of the code complexity. One addition was the inclusion of the `count_eq` constraint instead of a `sum` constraint to ensure that only one shift type is selected for every day, employee combination. We also experimented with the `indomain_min` and `largest` search annotation strategy, however, this lead to a degradation in performance.

F Model Improvements

Reducing model variables During the models development, after validating its correctness, various potential improvements were evaluated, some of which succeeded in improving the model runtime. The first thing we tried was to decrease the number of model variables. We did that by executing the "compile profiler" functionality of MiniZinc, which informed us about the number of variables produced by every constraint. In that way, we were able to replace many `forall` clauses with global constraints, like `exists, count, sum`. Having less variables of course improved the runtime. Namely, we observed a `30%` better performance when we substituted the `forall` predicate with global constraints in 3 of our constraints. This evaluation was conducted in all models for satisfaction objective.

Free search parameter One of the first things we realized while evaluating the model, was that the Chuffed solver had by far the best performance compared to the other backends. However, even after optimizing all the constraints to the best of our knowledge, we still were not able to solve all problem instances for satisfiability. Specifically, the largest instances `15, 19, 20` remained unsolvable after the 1-hour time-out. When we noticed the existence of the `free-search` parameter for Chuffed solver, we removed the specific search strategy we had set, to test the `free-search` strategy efficiency. The results were significantly better for the instances that were already solved, but we still couldn't get a feasible solution for the largest of the instances. We were able to solve all instances only after combining our specified search strategy with the `free-search` strategy. That is when we realized that Chuffed not always ignores the user-specified strategy. This is something we were not able to cross-validate based on the available literature online, but we are confident that the combination of the two strategies provides the optimal results in our problem.

Making offset a parameter At the beginning of the implementation, we had defined two variables; one was the array with the schedule of the first employee, and the other was the offset between the first day of the schedule and first day of the week (Monday). Because offset was a variable, a new symmetry was introduced, allowing the generated schedule to "shift" to different start days by assigning different values to the offset. For that reason, a constraint, specifying the value of the offset variable was introduced, initially as a symmetry breaking constraint. However, after validating the correctness of the model, we realized that there is no reason for the offset to be a variable, since its value is explicitly defined. Of course, the constraint which computed the offset became a normal constraint. This change improved the runtime of the largest instances by approximately 5%.

Custom modulo function The modulo operation is used repeatedly in our constraints. So, making a more efficient implementation was one of our model improvements. The code for the custom modulo function is given below:

```
% Function that performs calculations with modulus without using the
% explicit "mod" function.
function int: calc_mod(int: inp, int: divisor) =
  inp - bool2int(inp>divisor)*divisor + bool2int(inp<1)*divisor;
```

G Evaluation of Technologies and Backends

Based on the evaluation we performed with both models, we can confidently state that the most effective technology for the Rotating Workforce Problem (among the ones tested) is the Lazy Clause Generation (LCG), and namely the mzn-Chuffed backend. In the case of the first model, it was able to find a feasible solution for all 20 instances in under one minute. In the case of the maximization objective, it solved 12/20 problems to optimality, compared to Gecode solver, that only solved 8/20. Finally, Coin-BC solver performed worse than Chuffed in all instances checked. For model 2, we explored how a binary table design of representing the output affects performance. We came to the conclusion that it generally performs worse for this problem, probably because of the increased complexity of the code and the additional memory usage. Furthermore, it is also not computationally efficient because of the need to perform the modulo operations for 2 dimensions (days, employees). Similar to the first model, we saw the LCG-based backend, Chuffed perform better than the CP solver, Gecode. This is probably because LCG-backends work better for SAT-based constraints (which we tried to adopt as much as possible). In terms of results obtained, we observed that 17/20 examples were satisfied, and 12/20 were solved to optimality within the timeout period.

H Real Life Recommendation

In a real world application of the Rotating Workforce Scheduling problem, our team would suggest using the first model developed, with the LCG Technology and the Chuffed solver.
To begin with, Chuffed solver was proven capable of solving all provided instances to feasibility at least as fast as all other solvers. Its superiority can mainly be observed in the large instances, in which Chuffed could compute a feasible solution from the first minute of the execution. On the contrary the other tested solvers struggled with finding a solution to all instances before the time-out.

In terms of the model, we clearly observed that the first model, which only computed the schedule of the first employee, and used a rotating schedule, provided a better runtime for all the evaluated instances. This is expected, as this model has less variables.

Moreover, the modelling of the constraints of the first model was simpler in terms of understanding. This makes the first model more self-explanatory, and hence more easily-maintainable. That is, of course, an important aspect of a real-world application.

On the contrary, both models were built based on specific assumptions, which would limit possible alterations. The first assumption is that the ”forbidden sequences” feature only supports sequences of 2 or 3 shifts. In case the manager wanted to constraint shifts sequences of 4+ duration, new constraints should be introduced. In addition to that, we noticed that many constraints were built assuming the symmetry breaking constraints (first day of the schedule is a working day, last day is a day off) always hold. Hence, if for some reason, the aforementioned assumptions don't hold, multiple constraints of the model would need to be revisited.

Feedback to the Teaching Staff

Feedback from Konstantinos I personally enjoyed this project, because I had the chance to work on a new framework, and the flexibility to try virtually anything, and see ourselves whether it improves the model or not. I also enjoyed the fact that this was a group project, because in many phases of the project in-depth discussion was needed. Moreover, the project served as a great way to familiarize myself with MiniZinc.
On the other hand, I found the assignment questions a bit unclear. For instance, at first I had realized that we need to perform the whole extended version of the provided paper. Also, I found many questions being overlapping (f.i. Task A. Design and evaluate one model) and Task C. Experiment with the developed model). Fortunately, everything became clear after asking questions on Stackoverflow.

Feedback from Aditya On the negative side, I felt that a Windows-compatible Scala script could have been made available for us to easily test our models. The other annoying part was the updates made to the assignment description, due to which we had to modify our code. Overall, I found the assignment a good experience. I liked working on this assignment, especially because it helped me gain some practice in programming with MiniZinc. Another plus point is that it was a group project, so I learned a lot of new techniques when discussing with my teammate. The responses on the discussion forums were also very helpful.

Crash Report

All our experiments have been done with the free search parameter, -f, enabled. However, the automated Scala file that we received does not have the option to include this as an option. When running the automated tests for our model, we request you to run the models with the free search parameter enabled. Apart from this, there were no other problems w.r.t execution.

References

[1] Lucas Kletzander, Nysret Musliu, Johannes Gärtner, Thomas Krennwallner, and Werner Schafhauser. Exact methods for extended rotating workforce scheduling problems. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):519–527, May 2021.