

Maintenance scheduling

Linear and integer optimisation: Assignment 2

Victor Brun

May 12, 2022

The mathematical model

Sets and parameters

- \mathcal{N} = the set of components in the system.
- T - the number of time steps in the planning period.
- T_i - the life of a new component of type $i \in \mathcal{N}$ (measured in number of time steps). It is assumed that $2 \leq T_i \leq T - 1$.
- c_{it} - the cost of a spare component of type $i \in \mathcal{N}$ at time t (measured in €). For some instances it is assumed that c_{it} is constant over time, i.e., $c_{it} = c_i, t = 1, \dots, T$.
- d_t - the cost for a maintenance occasion at time t (measured in €). For some instances it is assumed that d_t is constant over time, i.e., $d_t = d, t = 1, \dots, T$.



Decision variables

$$x_{it} = \begin{cases} 1 & \text{if component } i \text{ is replaced at time } t, \\ 0 & \text{otherwise,} \end{cases} \quad i \in \mathcal{N}, t \in \{1, \dots, T\}.$$
$$z_t = \begin{cases} 1 & \text{if maintenance is made at time } t, \\ 0 & \text{otherwise,} \end{cases} \quad t \in \{1, \dots, T\}.$$

The model

$$\text{minimise} \quad \sum_{t=1}^T \left(\sum_{i \in \mathcal{N}} c_{it} x_{it} + d_t z_t \right), \quad (1)$$



$$\text{subject to} \quad \sum_{t=l+1}^{l+T_i} x_{it} \geq 1, \quad l = 0, \dots, T - T_i, i \in \mathcal{N}, \quad (2)$$



$$x_{it} \leq z_t, \quad t = 1, \dots, T, i \in \mathcal{N}, \quad (3)$$


$$x_{it}, z_t \in \{0, 1\}, \quad t = 1, \dots, T, i \in \mathcal{N}. \quad (4)$$


Problem 1

(a)

Decision var. cond.	$f(\mathbf{x}^*)$ [€]	Method	Solve time
$x_{it}, z_t \in \{0, 1\}$	762	Simplex	11.46 s
$x_{it} \in \{0, 1\}$	762	Simplex	15.93 s
$z_t \in \{0, 1\}$	762	Simplex	4.82 s
$x_{it}, z_t \in \mathbb{R}_+$	724	Dual simplex	0.05 s

Table 1: Solutions to the problem defined in `as_mod.jl` using the data in `as_dat_large.jl` with $T = 125$. Note that when only one decision variable is present in the column *Decision var. cond.* it means that the other variable lies within \mathbb{R}_+ .

By observing the different solutions in table 1 that correspond to different conditions on the decision variables, we can see that only one solution gives a different optimal objective value, namely the solution corresponding to $x_{it}, z_t \in \mathbb{R}_+$. To understand this we need interpret what the relaxations mean. When relaxing the integrality constraint on x_{it} it becomes possible to change only a portion of the the component i at time t , and when the same constraint is relaxed for z_t the maintenance occasions can be partial. However, for the case $x_{it} \in \mathbb{R}_+, z_t \in \{0, 1\}$ the solution becomes integer in x_{it} anyway, since constraint (2) forces the whole component i to be changed at the maintenance occasion t which is still binary. 

The same behaviour is observed for the case $x_{it} \in \{0, 1\}, z_t \in \mathbb{R}_+$ and is explained by constraint (3) which forces z_t to be binary since the integrality constraint on x_{it} means that, at every maintenance occasion, the whole component must be changed. Furthermore, this binary forcing behaviour explains why the optimal value for the non-relaxed problem is the same as for the partially relaxed problems, which can be observed in table 1. 

In table 1 we can also observe that the solution to the fully relaxed problem gives a smaller optimal value. This can be explained by the fact that it is possible to change a component later in its lifespan without increasing the cost of maintenance occasions too much. Mathematically, we can also say that the relaxation increased the feasible region in the direction of the optimum, thus yielding a better solution.

In table 1, the computation times for the different solutions are also presented. As expected the ILP model is computationally more cumbersome than the LP model, which is due to how the ILP model is solved. When solving an ILP model one usually relaxes it to an LP model, and uses the solution to that model and searches for the best integer solution using something like The cutting planes algorithm. What however can be surprising is; 1. the spread of the computation time for the MILP models, and 2. that the MILP model with $x_{it} \in \{0, 1\}$ and $z_t \in \mathbb{R}_+$ is the, computationally, hardest problem.

(b)

Solving the ILP problem as specified in `as_run.jl` using the data in `as_dat_small.jl` yields:

$$f(X^*, \mathbf{z}^*) = 14, \quad X^* = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{z}^* = (0 \ 0 \ 1 \ 1)^T. \quad (5)$$

Solving the ILP problem that is created when relaxing the integrality constraints in above LP yields:

$$f(X^*, \mathbf{z}^*) = 13.5, \quad X^* = \frac{1}{2} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{z}^* = \frac{1}{2}(1 \ 0 \ 1 \ 1)^T. \quad (6)$$

Solving the above ILP problem, but with the extra constraint a

$$z_1 + x_{12} + x_{22} + x_{13} + x_{23} + z_4 \geq 2, \quad (7)$$

yields:

$$f(X^*, \mathbf{z}^*) = 13.5, \quad X^* = \frac{1}{2} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{z}^* = \frac{1}{2}(1 \ 0 \ 1 \ 1)^T. \quad (8)$$

The difference between (5) and (6) implies that relaxing the integrality constraints expands the feasible region in the direction of the optimal solution. Moreover, the difference, or more accurately non-difference, between (6) and (8) implies that the added constraint, (7), does not reduce the feasible region in the direction of the optimal solution.



We wish to prove that by adding the constraint (7) we do not alter the feasible region generated by (2), (3), (4). By using the data in `as_dat_small.js` we can expand 2 to

$$x_{11} + x_{12} + x_{13} \geq 1 \quad (9)$$

$$x_{12} + x_{13} + x_{14} \geq 1 \quad (10)$$

$$x_{21} + x_{22} + x_{23} + x_{24} \geq 1. \quad (11)$$

Adding (9) and (10) yields

$$x_{11} + 2x_{12} + 2x_{13} + x_{14} \geq 2.$$

Now, since $x_{it} \geq 0$ we can add x_{it} terms to (11) without breaking the inequality, thus

$$x_{21} + 2x_{22} + 2x_{23} + x_{24} \geq 1.$$

Furthermore, constraint (3) makes it possible to remove x_{it} with z_t , giving

$$z_1 + 2x_{12} + 2x_{13} + z_4 \geq 2$$

$$z_1 + 2x_{22} + 2x_{23} + z_4 \geq 1.$$

Lastly, by adding above constraints and dividing by two we arrive at

$$z_1 + x_{12} + x_{22} + x_{13} + x_{23} + z_4 \geq \frac{3}{2}.$$

Since this inequality is derived from the original constraints and since it is more restrictive than (7), the feasible region will not change by adding (7).

Problem 2

(a)

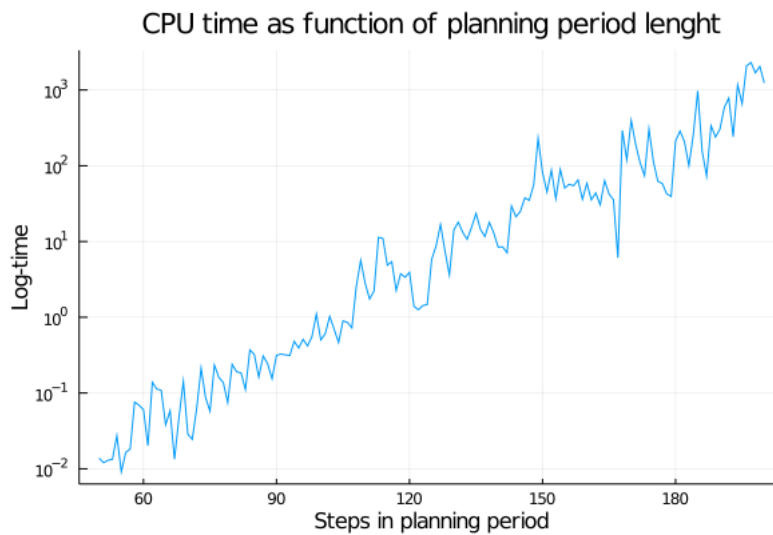



Figure 1: Solution time, in log time scale, for the maintenance scheduling model with relaxed integrality condition on x_{it} plotted against T . The MIPGap is set to 1%. 

(b)

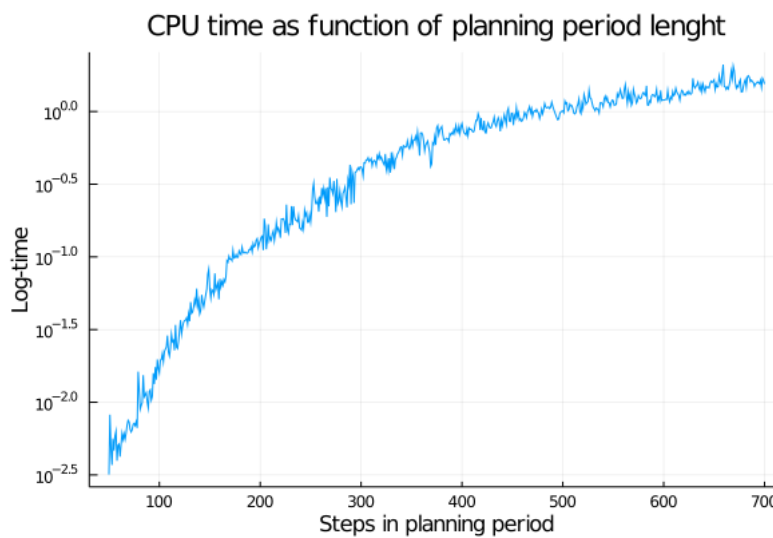


Figure 2: Solution time, in log time scale, for the maintenance scheduling model with relaxed integrality condition on both x_{it} and z_t , plotted against T . The MIPGap is set to 1%.

(c)

Figure 1 shows that the solution time, in log scale, appears to be linear, which indicates that the time complexity of the implemented solver is exponential. This is expected since MILP problems are NP-hard. Furthermore, by observing figure 2 it can be seen that the solution time, in log scale, flattens out, which indicates that the time complexity of the implemented solver is polynomial. This is expected, as well, since without any integrality constraints on x_{it} and z_t the problem reduces to an LP problem, which indeed are solvable in polynomial time.

(d)

For the case where the integrality constraint on x_{it} is relaxed, the solver spends little time on presolve, less than 0.1 s for each solution. The time spend on finding the optimal solution is a bit more, but it generally takes the solver a few seconds. However, in some cases, the solver finds the optimal solution late in the optimality verification of a suboptimal solution. When comparing this time to the time it takes to verify a solutions optimality, it is however short. The solver can spend several minutes verifying a solutions optimality, which is not surprising since many branches need to be searched in order to verify that a solution is optimal, while the optimal solution only need to be found once.

For the case where our model has no integrality constraints there is one big difference. Due to the relaxation we are left with an LP problem, and the optimal solution can thus be found without the use of the branch-and-bound algorithm. This dramatically lowers the time spent on verifying an optimal solution.

Problem 3

(a)

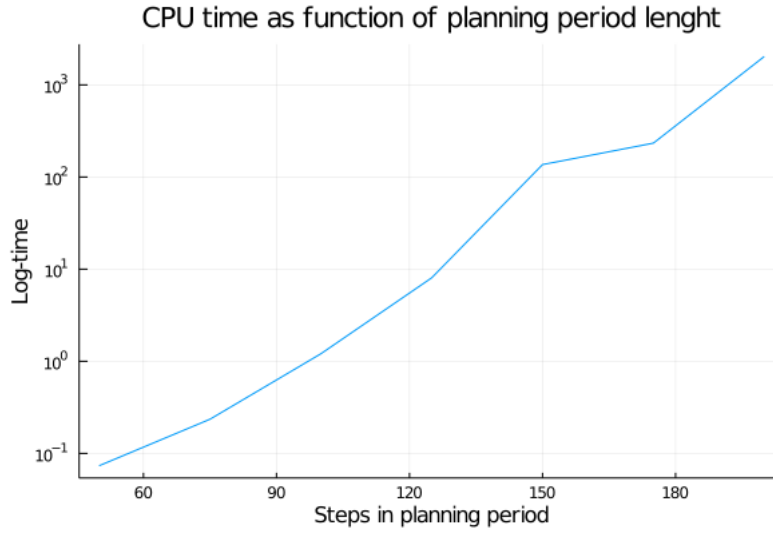



Figure 3: Solution time, in log time scale, for the generalised maintenance scheduling model with relaxed integrality condition on x_{st}^i plotted against T . The MIPGap is set to 3%. 

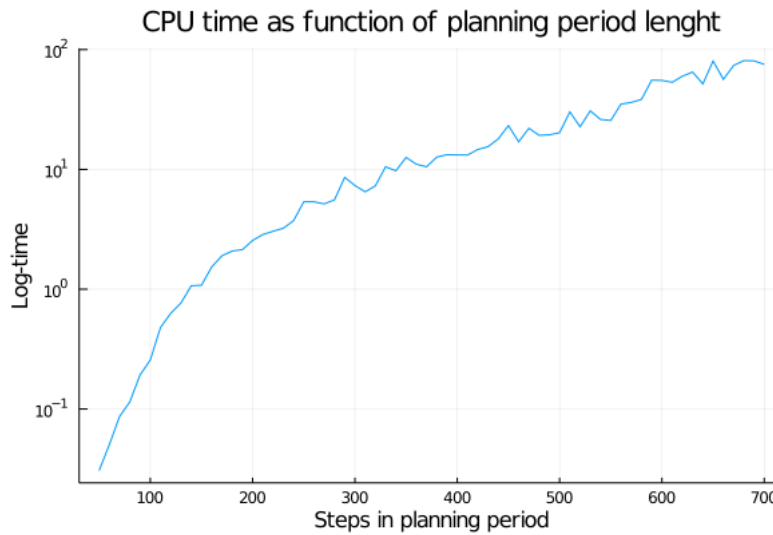



Figure 4: Solution time, in log time scale, for the generalised maintenance scheduling model with relaxed integrality condition on both x_{st}^i and z_t , plotted against T . The MIPGap is set to 2%. 


(b)

Figure 3 is very similar to figure 1 in that it shows a linear relationship between the planning period length and the log-time. As discussed above, this indicates that

the implemented solver for the generalisation model, presented in *E. Gustavsson et al., 2014*, has exponential time complexity, when only the integrality constraint on x_{st}^i is relaxed. Furthermore, comparing figure 3 with figure 1 it can be seen that for a given T the solution time is approximately the same in both cases, this means that the generalised model has a somewhat slower solution time since it is solved with a higher MIPGap. This is not surprising since it introduces an extra dimension to the x -variable compared to our model.

Regarding figure 4 it can be seen having the same flattening behaviour as figure 2, which indicates that the implemented solver has polynomial time complexity. Comparing the figures we can further more see that the solution time for the generalised model is slower than for our model, which again is not surprising since the generalised model introduces an extra dimension to the x -variable.

(c)  

The main difference between our model ((1), (2), (3), (4)) and its generalisation presented in *E. Gustavsson et al., 2014* is the extra dimension introduced into the x -variable and subsequently also the c -parameter. By adding the extra dimension,  the x -variable no longer defines the maintenance occasion and the c -parameter no longer defines the replacement cost, instead they define the interval between maintenance occasions and interval cost, respectively.

This difference, or generalisation, is what is needed to convert our problem into a network flow problem. By considering the maintenance occasions z_t as the existing nodes in a network, the maintenance interval x_{st}^i are the arcs. The costs of adding a new node is d_t while the cost of adding a new arc is c_{st}^i . The problem thus becomes to choose nodes and arcs such that our conditions are met while minimising the total cost of the network.

Problem 4

(a)

In order for a given component i to have a remaining life of $r > 0$ at the end of the planning period, T , it must have been replaced at least one during the period $[T - T_i + r, T]$. Expressing this as a condition we get that

$$\sum_{t=T-T_i+r}^T x_{it} \geq 1, \quad i \in \mathcal{N}.$$

By adding above constraint, see `RemainingLife` in B.2, to the model in `as_mod.jl` and running it with $d = 20$, $T = 100$, $r = 10$, we can verify that the constraint actually works by observing that

$$\mathbf{x}[2, T-5:T] = [-0.0, -0.0, -0.0, -0.0, 1.0, 0.0].$$

Since the life time of component $i = 2$ is $T_2 = 11$, it must be changed at $T - 1$ or T in order for it to have $r \geq 10$ time steps of life time left after T , which is exactly what we can see above where it is replaced at $T - 1$.

(b) & (c)

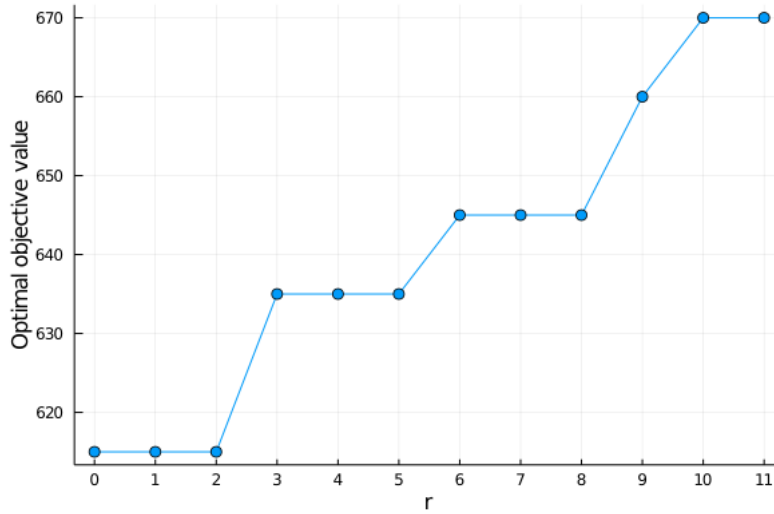


Figure 5: Optimal objective value plotted against the number of minimum required time steps, r to have left of life time for every component at end of planning period, T . No optimal objective value exists for $r > 11$ since the problem becomes infeasible.

In figure 5 we see that the optimal objective value only changes at $r = 3, 6, 9, 10$. Moreover, there exists no r -values larger than 11 for our problem since it becomes infeasible because the shortest life span of a component is $T_{10} = 11$.

In table 2 we present the five different optimal objective value levels, $r = 3, 6, 9, 10$, and a sixth level $r = 2$ which has a different number of maintenance occasions, in more detail. As can be seen in the table, the cheapest solution can be found for $r \leq 2$. Furthermore, at $r = 2$ the objective stays the same while the number of maintenance occasions and components replaced changes, which means that in order for the minimum number of time steps left of every components life time at the end of the planning period to be 2, some the maintenance occasions need to be more frequent, but the number of components changed during each occasion is fewer, compared to $r = 0$. For $r = 3$ and 6, the number of components replaced increase while maintenance occasions stay the same, which is an expected behaviour. However, for $r = 9$, the number of occasions and components replaced is the same as for $r = 6$, but with a higher objective value. This means that a more expensive component needs to be replaced late in the planning period while a less expensive component has been changed sufficiently late in the planning period to satisfy the added condition. For $r = 10$ and 11 the same behaviour as for $r = 3$ and 6 can be seen, and for $r > 11$ the problem becomes, as mentioned above, unfeasible.

r	$f(\mathbf{x}^*)$	# maintenance occasions	# components replaced
0	615	10	29
2	615	11	27
3	635	11	29
6	645	11	30
9	660	11	30
10	670	11	31

Table 2: Table over how the optimal objective value, number of maintenance occasions and components replaced changes with the minimum number of time steps, r , that each component must have left of their life time at the end of the planning period. For the r -values that are not present in the table, they have the same objective values, number of maintenance occasions and components replaced changes as the respective before-coming r -value.

A Julia code for problem 3

A.1 as_ext_dat_large.jl

```

1 # Sets
2 Components = 1:10 # 10 components
3
4 # Parameters
5 d = ones(1,T+1)*20 #cost of a maintenance occasion
6 d[1,T+1] = 0 # As spec. in article this is no maintenance occasion so
   ↪ cost is zero
7 c = [34 25 14 21 16 3 10 5 7 10]*ones(1,T+1) #costs of new
   ↪ components
8 c[:,T+1] = zeros(10,1) # As spec. in article this is no maintenance
   ↪ occasion so cost is zero
9 U = [42 18 90 94 49 49 34 90 37 11] #lives of new components

```

A.2 as_ext_mod.jl

```

1 function build_extended_model(T::Int, relax_x::Bool, relax_z::Bool)
2     m = Model()
3
4     #  $x(i,t,s) = 1$  if component  $i$  is replaced at time  $s$ 
5     # and time  $t$ , but not inbetween, otherwise 0.
6     if relax_x
7         @variable(m, x[i in Components, t in 1:T+1, s in 0:t-1] >= 0)
8     else
9         @variable(m, x[i in Components, t in 1:T+1, s in 0:t-1] >= 0,
           ↪ Bin)

```

```

10     end
11
12     # 1 if maintenance is allowed at time t,
13     # otherwise 0.
14     if relax_z
15         @variable(m, z[1:T+1] >= 0)
16     else
17         @variable(m, z[1:T+1] >= 0, Bin)
18     end
19
20     # Cost functions
21     cost = @objective(
22         m, Min, sum(d[t]*z[t] for t in 1:T) +
23         sum(c_fn(i,t,s)*x[i,t,s] for i in Components, t in 1:T+1, s in
24             ↪ 0:t-1)
25     )
26
27     # Make sure that maintenance can only performed when z is one
28     ReplaceOnlyAtMaintenance = @constraint(
29         m,
30         [i in Components, t in 1:T],
31         sum(x[i,t,s] for s in 0:t-1) <= z[t]
32     )
33
34     # for each component i, ensure that the same
35     # number of maintenance intervals end and start at time t
36     StartEndSameNumber = @constraint(
37         m,
38         [i in Components, t in 1:T],
39         sum(x[i,t,s] for s in 0:t-1) == sum(x[i,r,t] for r in t+1:T+1)
40     )
41
42     # ensure that maintenance interval for component i
43     # starts at time t = 0.
44     StartMaintenanceIntervalAtZero = @constraint(
45         m,
46         [i in Components],
47         sum(x[i,t,0] for t in 1:T+1) == 1
48     )
49
50     return m, x, z
51 end
52
53 # In order for the model to be penalised for choosing
54 # a service gap, t-s, larger than the life time of a given
55 # component, i, we modify the service cost to be large when
56 # t-s >= U[i], otherwise we just return the cost of component
57 # i at time t.
58 function c_fn(i::Int ,t::Int ,s::Int)
59     if t-s > U[i]

```

```
59         return 10^5
60     else
61         return c[i,t]
62     end
63 end
```

A.3 as_dat_large.jl

```
1 using JuMP
2 using Gurobi
3 using SparseArrays
4 using Plots
5
6 T_min = 50
7 T_max = 200
8 step_size = 10
9 relax_x = true
10 relax_z = false
11
12 solve_times_vec = []
13 for ix in T_min:step_size:T_max
14     global T = ix
15     print("T = ", T, "\n")
16
17     # We include this here in order for the T-dependent
18     # parameters to be created correctly
19     include("as_ext_mod.jl")
20     include("as_ext_dat_large.jl")
21
22
23     # Builds model and sets optimiser
24     m, x, z = build_extended_model(T, relax_x, relax_z)
25     set_optimizer(m, Gurobi.Optimizer)
26     set_optimizer_attributes(m, "MIPGap" => 3e-2, "TimeLimit" => 3600)
27
28     # Solves optimisation problem
29     optimize!(m)
30
31     t = solve_time(m)
32     append!(solve_times_vec, t)
33 end
34
35 plot(T_min:step_size:T_max, solve_times_vec, yaxis=:log,
36      title="CPU time as function of planning period lenght",
37      legend=false)
38 xlabel!("Steps in planning period")
39 ylabel!("Log-time")
```

B Julia code for problem 4

B.1 as_dat_large.jl

```

1 # Sets
2 Components = 1:10 # 10 components
3
4 # Parameters
5 T = 100 #number of timesteps
6 # r = 10
7 d = ones(1,T)*20 #cost of a maintenance occasion
8 c = [34 25 14 21 16 3 10 5 7 10]*ones(1,T) #costs of new
   ↪ components
9 U = [42 18 90 94 49 49 34 90 37 11] #lives of new components

```

B.2 as_mod.jl

```

1 """
2 Construct and returns the model of this assignment.
3 """
4 function build_model(relax_x::Bool = false, relax_z::Bool = false)
5 #Components - the set of components
6 #T - the number of time steps in the model
7 #d[1,...,T] - cost of a maintenance occasion
8 #c[Components, 1,...,T] - costs of new components
9 #U[Components] - lives of new components
10 m = Model()
11 if relax_x
12     @variable(m, x[Components, 1:T] >= 0)
13 else
14     @variable(m, x[Components, 1:T] >= 0, Bin)
15 end
16 if relax_z
17     @variable(m, z[1:T] <= 1)
18 else
19     @variable(m, z[1:T] <= 1, Bin)
20 end
21 cost = @objective(m, Min,
22     sum(c[i, t]*x[i, t] for i in Components, t in 1:T) + sum(d[t]*z[t]
   ↪ for t in 1:T))
23
24 ReplaceWithinLife = @constraint(m,
25     [i in Components, ell in 0:(T-U[i]); T >= U[i]],
26     sum(x[i,t] for t in (ell .+ (1:U[i]))) >= 1)
27
28 ReplaceOnlyAtMaintenance = @constraint(m, [i in Components, t in
   ↪ 1:T],

```

```

29  x[i,t] <= z[t])
30
31  RemainingLife = @constraint(
32      m,
33      [i in Components],
34      sum(x[i,t] for t in (T-U[i]+r):T) >= 1
35  )
36
37  return m, x, z
38 end
39 """
40  Adds the constraint:  z[1] + x[1,2] + x[2,2] + x[1,3] + x[2,3] + z[4]
    ↪ >= 2
41  which is a VI for the small instance
42 """
43 function add_cut_to_small(m::Model)
44     @constraint(m, z[1] + x[1,2] + x[2,2] + x[1,3] + x[2,3] + z[4] >= 2)
45 end

```

B.3 as_run.jl

```

1  using JuMP
2  #using Cbc
3  using Gurobi
4  using SparseArrays
5  using Plots
6
7  r_min = 0
8  r_max = 11
9  objective_values = []
10 z_sums = []
11 x_sums = []
12 for ix in r_min:r_max
13     global r = ix
14
15     # We include this here in order for the r-dependent
16     # parameters to be created correctly
17     include("as_mod.jl")
18     include("as_dat_large.jl")
19
20     # Builds model and sets optimiser
21     m, x, z = build_model()
22     set_optimizer(m, Gurobi.Optimizer)
23
24     # Optimises the problem and saves the
25     # bjective value
26     optimize!(m)
27     append!(objective_values, objective_value(m))

```

```
28
29     z_sum = sum(value.(z)[t] for t in 1:T)
30     x_sum = sum( sum(value.(x.data)[i,t] for t in 1:T) for i in
    ↪ Components )
31     append!(z_sums, z_sum)
32     append!(x_sums, x_sum)
33 end
34
35 plot(r_min:r_max, objective_values,
36      marker=(:circle,5),
37      xticks=r_min:r_max,
38      legend=false)
39 xlabel!("r")
40 ylabel!("Optimal objective value")
41 print("optimal objective values: ", objective_values, "\n")
42 print("# maintenance occasions: ", z_sums, "\n")
43 print("# components replaced: ", x_sums, "\n")
```
