Danmarks Tekniske Universitet

DTU

42136 - Large Scale Optimization Using Decomposition

Authors

Group 25
Karen Witness - s196140
Aksel Buur - s203947

May 22, 2025

Original CLSP model

$$\min_{n,m} \quad \sum_{i=1}^{n}\sum_{t=1}^{m}(c_i y_{it} + h_i s_{it}) \tag{1}$$

$$\text{s.t.} \quad x_{i1} = d_{i1} + s_{i1} \quad \forall i = 1,\dots,n \tag{2}$$

$$s_{i,t-1} + x_{it} = d_{it} + s_{it} \quad \forall i = 1,\dots,n,\ t = 2,\dots,m \tag{3}$$

$$x_{it} \leq M_1 y_{it} \quad \forall i = 1,\dots,n,\ t = 1,\dots,m \tag{4}$$

$$\sum_{i=1}^{n} x_{it} + q_i y_{it} \leq C \quad \forall t = 1,\dots,m \tag{5}$$

$$y_{it} \in \{0,1\} \quad \forall i = 1,\dots,n,\ t = 1,\dots,m \tag{6}$$

$$x_{it} \in \{0,1,2,\dots,M_2\} \quad \forall i = 1,\dots,n,\ t = 1,\dots,m \tag{7}$$

$$s_{it} \in \{0,1,2,\dots,M_3\} \quad \forall i = 1,\dots,n,\ t = 1,\dots,m \tag{8}$$

Task 1

The objective function minimizes the total cost of setup, production, and inventory of all products in all time periods. Each product have a setup cost c which is added to the cost function one time if the production of the product i is setup in time period i. For each product i that is in inventory at the end of period t, a holding cost must be payed every time.

- Demand constraint (2) is ensuring that each product in the first time period will produce as much as the demand requires in the first time period and inventory is updated.
- Inventory constraint (3) is updating inventory for each product in every time period.
- Turn on/off constraint (4) ensures that each product can only be produced if the production is setup.
- Capacity constraint (5) ensures the sum of produced products x and production "units" if the setup is turned on, doesn't exceed the limit of production.

We can compute the values for M1 and M3 by assuming that x cannot be than more than max of demand of product 1 or product 2. To compute the value for M2, we compare the minimum of the value of the total demand for each product and the capacity for each product.

Product i	a_i $\sum_t d_{it}$	b_i $C - q_i$	$\min(a_i, b_i)$
1	12	8	8
2	7	9	7

Table 1: Computing values for M_1, M_2, M_3

From table (1) where we have computed the total demand and capacity for each product, we determine M-values to be the following.

$$M_1 = \max(a_1, a_2) = \max(12, 7) = 12$$
$$M_2 = \max(\min(a_1, b_1), \min(a_2, b_2)) = \max(8, 7) = 8$$
$$M_3 = \max(a_1, a_2) = \max(12, 7) = 12$$

Task 2

In table (2) we see the constraint matrix A. All four constraints is written for each product and each time period which result in 15 rows of constraints and 18 columns of variables. The constraints are rearranged so all the variables with coefficients are on the left side and b standing alone on the right-hand side.

Constraint matrix

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}	y_{11}	y_{12}	y_{13}	y_{21}	y_{22}	y_{23}	s_{11}	s_{12}	s_{13}	s_{21}	s_{22}	s_{23}	$\leq/=$	RHS	Cons.#
1	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	=	3	2
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	=	0	2
3	0	1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	=	4	3
4	0	0	1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	=	5	3
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	-1	0	=	5	3
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	-1	=	2	3
7	1	0	0	0	0	0	-12	0	0	0	0	0	0	0	0	0	0	0	$\leq$	0	4
8	0	1	0	0	0	0	0	-12	0	0	0	0	0	0	0	0	0	0	$\leq$	0	4
9	0	0	1	0	0	0	0	0	-12	0	0	0	0	0	0	0	0	0	$\leq$	0	4
10	0	0	0	1	0	0	0	0	0	-12	0	0	0	0	0	0	0	0	$\leq$	0	4
11	0	0	0	0	1	0	0	0	0	0	-12	0	0	0	0	0	0	0	$\leq$	0	4
12	0	0	0	0	0	1	0	0	0	0	0	-12	0	0	0	0	0	0	$\leq$	0	4
13	1	0	0	1	0	0	2	0	0	1	0	0	0	0	0	0	0	0	$\leq$	10	5
14	0	1	0	0	1	0	0	2	0	0	1	0	0	0	0	0	0	0	$\leq$	10	5
15	0	0	1	0	0	1	0	0	2	0	0	1	0	0	0	0	0	0	$\leq$	10	5

Table 2: Constraint matrix A, right-hand side vector b, and constraint types for the CLSP model.

Task 3

We reorder the matrix 2, for product 1 and product 2 separately. This shows that for each product, all constraints are coupled, meaning we will have $n = 2$ independent subproblems.

Constraint matrix ordered by products

	Product 1									Product 2									≤/=	RHS	Cons.#
	x_{11}	x_{12}	x_{13}	y_{11}	y_{12}	y_{13}	s_{11}	s_{12}	s_{13}	x_{21}	x_{22}	x_{23}	y_{21}	y_{22}	y_{23}	s_{21}	s_{22}	s_{23}			
1	1	0	0	0	0	0	−1	0	0	0	0	0	0	0	0	0	0	0	=	3	2
2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	−1	0	0	=	0	2
3	0	1	0	0	0	0	1	−1	0	0	0	0	0	0	0	0	0	0	=	4	3
4	0	0	1	0	0	0	0	1	−1	0	0	0	0	0	0	0	0	0	=	5	3
5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−1	0	=	5	3
6	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−1	=	2	3
7	1	0	0	−12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	≤	0	4
8	0	1	0	0	−12	0	0	0	0	0	0	0	0	0	0	0	0	0	≤	0	4
9	0	0	1	0	0	−12	0	0	0	0	0	0	0	0	0	0	0	0	≤	0	4
10	0	0	0	0	0	0	0	0	0	1	0	0	−12	0	0	0	0	0	≤	0	4
11	0	0	0	0	0	0	0	0	0	0	1	0	0	−12	0	0	0	0	≤	0	4
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	−12	0	0	0	≤	0	4
13	1	0	0	2	0	0	0	0	0	1	0	0	1	0	0	0	0	0	≤	10	5
14	0	1	0	0	2	0	0	0	0	0	1	0	0	1	0	0	0	0	≤	10	5
15	0	0	1	0	0	2	0	0	0	0	0	1	0	0	1	0	0	0	≤	10	5

Table 3: Reordered constraint matrix A grouped by product (first 9 columns for product 1, next 9 for product 2), with RHS and constraint types.

Task 4

From the table in task 3 we are able to reformulate the CLSP by using Dantzig-Wolfe decomposition. We introduce for each product i, a set of all feasible production patterns, Ω_i, over the planning horizon. Ω_i is typically an exponential growing set.
We create the master problem using the standard Dantzig-Wolfe choice of variables, where we have convexified the capacity constraint. The master problem minimizes the total cost of setup and storage over the integer patterns in the set Ω_i.
Master problem:

$$\min_{p\in\Omega} \quad \sum_{i=1}^{n}\sum_{p\in\Omega_i}(c_i y_{ip} + h_i s_{ip})\,\lambda_{ip} \tag{9}$$

$$\text{subject to} \quad \sum_{i=1}^{n}\sum_{p\in\Omega_i}(x_{it} + q_i y_{it})\lambda_{ip} \le C \quad \forall t = 1,\dots,m \quad (\pi) \tag{10}$$

$$\sum_{p\in\Omega_i}\lambda_{ip} = 1 \quad \forall i = 1,2 \quad (\kappa) \tag{11}$$

$$\lambda_{ip} \ge 0 \tag{12}$$

We also denote the dual variables here as π for the capacity constraint and κ for the weights of λ. Each column added, p, for product i will contribute to the objective of the master problem.

The subproblems will for each product i generate a least-reduced-cost pattern by solving the the problem subject to the remaining constraints from the CLSP. By enforcing $y_{it} \in \{0,1\}$ and having x_{it}, s_{it} continuous, we exploit the network's uni-modularity to obtain integer

production and integer inventory levels without having to spend computation on forcing integrality. This will generate feasible columns and a bound strictly stronger than the LP-relaxation.

Subproblem product 1:

$$\min_{x,y,s} \quad \sum_{t=1}^{m} ((c_1 - \pi_t q_1) y_{1t} + h_1 s_{1t} - \pi_t x_{1t}) - \kappa_1 \tag{13}$$

$$\text{subject to} \quad x_{11} - s_{11} = d_{11} \tag{14}$$

$$s_{1t-1} + x_{1t} - s_{1t} = d_{1t}, \quad \forall t = 2, 3 \tag{15}$$

$$x_{1t} \leq M_1 y_{1t}, \quad \forall t = 1, \ldots, m \tag{16}$$

$$x_{1t}, s_{1t} \geq 0, \quad \forall t = 1, \ldots, m \tag{17}$$

$$y_{1t} \in \{0, 1\} \tag{18}$$

Subproblem product 2:

$$\min_{x,y,s} \quad \sum_{t=1}^{m} ((c_2 - \pi_t q_2) y_{2t} + h_2 s_{2t} - \pi_t x_{2t}) - \kappa_2 \tag{19}$$

$$\text{subject to} \quad x_{21} - s_{21} = d_{21} \tag{20}$$

$$s_{2t-1} + x_{2t} - s_{2t} = d_{2t}, \quad \forall t = 2, 3 \tag{21}$$

$$x_{2t} \leq M_1 y_{2t}, \quad \forall t = 1, \ldots, m \tag{22}$$

$$x_{2t}, s_{2t} \geq 0, \quad \forall t = 1, \ldots, m \tag{23}$$

$$y_{2t} \in \{0, 1\} \tag{24}$$

To solve this, using the Dantzig-Wolfe Decomposition, its is from Task 2 and Task 3, that independent subproblems can be used. This means that every time the subproblem is solved, both subproblems are solved resulting in new variables for the restricted master problem.
In julia, the master problem is implemented as follows for the first iteration. The code both for the master problem and sub-problem are greatly inspired by the exercises and solutions from week 3 in the Dantzig-Wolfe lectures, credits to Stefan Røpke for providing the code-framework.
It is chosen to initialize the extreme points in the first iteration for product 1 as:

$$X[1] = [7,\ 0,\ 5,\ 1,\ 0,\ 1,\ 4,\ 0,\ 0]$$

corresponding to the variables in table 3, meaning in the first period 7 units is produced ($x_{11} = 7$), covering the products needed for both period $t = 1$ and period $t = 2$. The last products for period $t = 3$ is produced at $t = 3$, meaning $x_{13} = 5$. This results in the remaining values: $y_{11} = 1,\ y_{12} = 0,\ y_{13} = 1,\ s_{11} = 4,\ s_{12} = s_{13} = 0$. In the same way, for product 2 the first iteration extreme points is chosen as:

$$X[2] = [0,\ 5,\ 2,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0]$$

Where it can be seen that the reason for producing all demand for product 1 in time period 1 is to not have an infeasible time period $t = 2$, since it would not be possible to just cover the all demand for both products in the time periods it is needed.
With this initial extreme points, which can be any feasible solution, it is possible to start the Dantzig-Wolfe Decomposition.

Listing 1: Data definitions

```
# costs and times
c1, c2 = 1.0, 2.0
h1, h2 = 1.0, 1.0
# subproblem cost vectors (length 9: [x]-coeffs zero; then y;
   then s)
c_sub1 = [0,0,0, c1,c1,c1, h1,h1,h1]
c_sub2 = [0,0,0, c2,c2,c2, h2,h2,h2]
c_full  = vcat(c_sub1, c_sub2)

# demands
d = [3,4,5,  0,5,2]  # first 3 for product1, next 3 for product2
#display(d)

A = [
    # flow eq & inventory linking for both products
    1 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   1 0 0   0 0 0  -1 0 0;
    0 1 0   0 0 0   1 -1 0   0 0 0   0 0 0   0 0 0;
    0 0 1   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 1 0   0 0 0   1 -1 0;
    0 0 0   0 0 0   0 0 0   0 0 1   0 0 0   0 1 -1;
    # big-M
    1 0 0  -12 0 0   0 0 0   0 0 0   0 0 0   0 0 0;
    0 1 0   0 -12 0  0 0 0   0 0 0   0 0 0   0 0 0;
    0 0 1   0 0 -12  0 0 0   0 0 0   0 0 0   0 0 0;
    0 0 0   0 0 0   0 0 0   1 0 0  -12 0 0   0 0 0;
    0 0 0   0 0 0   0 0 0   0 1 0   0 -12 0   0 0 0;
    0 0 0   0 0 0   0 0 0   0 0 1   0 0 -12  0 0 0;
    # capacity <= 10 in each period
    1 0 0   2 0 0   0 0 0   1 0 0   1 0 0   0 0 0;
    0 1 0   0 2 0   0 0 0   0 1 0   0 1 0   0 0 0;
    0 0 1   0 0 2   0 0 0   0 0 1   0 0 1   0 0 0
]
b = [3, 0, 4, 5, 5, 2, 0,0,0,0,0,0, 10,10,10]
# master slices
A0 = A[13:15, :]
b0 = b[13:15]
```

```
#println(b0)
# number of subproblems
K = 2

# columns in each subproblem
V = [
    1:9,      # product 1 uses vars 1-9
    10:18     # product 2 uses vars 10-18
]

# row-blocks for each subproblem: first the 3 demand eqns, then the 3 big-M ineqns
eq   = [[1,3,4],   [2,5,6]]
ineq = [[7,8,9], [10,11,12]]
subBlocks = [
  vcat(eq[1],   ineq[1]),
  vcat(eq[2],   ineq[2])
]

# pre-allocate container types
CV    = Vector{Vector{Float64}}(undef, K)
A_V   = Vector{Matrix{Float64}}(undef, K)
A0_V  = Vector{Matrix{Float64}}(undef, K)
b_sub = Vector{Vector{Float64}}(undef, K)

for k in 1:K
    # 1) the cost-vector for sub-problem k
    CV[k]   = c_full[V[k]]
    # 2) its full A1 (6x9) and b1 (length 6)
    A_V[k]  = vcat(
                  A[ eq[k],    V[k] ],
                  A[ ineq[k], V[k] ]
              )
    b_sub[k] = vcat(
                  b[ eq[k] ],      # the three demand RHS
                  zeros(length(ineq[k]))  # the three big-M zeros
              )
    # 3) the capacity slice (3x9)
    A0_V[k] = A0[:, V[k]]
end
```

This data-script is now used for solving the master problem(s) and subproblems, where it can be seen imported using the "include("task4-Data.jl")" line.

Listing 2: Iteration 1 - master problem

```
module test

using JuMP, GLPK, LinearAlgebra
include("task4-Data.jl")

# replace the following with your actual initialization:
X = Vector{Matrix{Float64}}(undef, K)
X[1] = [7 0 5 1 0 1 4 0 0]'   # one seed-column for product 1
X[2] = [0 5 2 0 1 1 0 0 0]'  # one seed-column for product 2
P = [1,1]                 # P[k] number of extreme points for
   polyhedron k
println("X[1] = ", X[1])
println("X[2] = ", X[2])
println("P = ",P)

# build the master
master = Model(GLPK.Optimizer)

# a separate lambda-array for each product
@variable(master, lambda1[1:P[1]] >= 0)
@variable(master, lambda2[1:P[2]] >= 0)
lambda = [lambda1, lambda2]
# objective: sum_i CV[i]'*X[i]*lambda[i]
@objective(master, Min,
    sum( CV[i]' * X[i] * lambda[i] for i in 1:K )
)

# capacity coupling: A0_V[i]*X[i]*lambda[i] summed over i <= b0
cons = Vector{ConstraintRef}(undef, 3)
for t in 1:3
  cons[t] = @constraint(master,
    sum(A0_V[i]*X[i]*lambda[i] for i=1:K)[t] <= b0[t]
  )
end
# convexity: for each i, sum_j lambda[i][j] == 1
@constraint(master, conv[i=1:K],
    sum(lambda[i][j] for j in 1:P[i]) == 1
)

optimize!(master)

if termination_status(master) == MOI.OPTIMAL
    println("Master Obj = ", objective_value(master))
```

```
    lambda-val = [value.(lambda[i]) for i in 1:K]
    println("lambda-val = ", lambda-val)
    pi = [ dual(cons[i]) for i in 1:3 ]
    kappa = [ dual(conv[i]) for i in 1:K ]
    println("pi = ", pi, "  kappa = ", kappa)
else
    error("Master not optimal: ", termination_status(master))
end

end
```

Other notes to the code is, that the matrices "A0_V" and "CV" is constructed in the file `task4-Data.jl`, see listing 1, as the constraints that are kept in the master problem from table 3 and "CV" is the original objective function for both products meaning:

$$\text{CV} = [0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 2, 2, 1, 1, 1]$$

Where the order of the variables is the same as in table 3, meaning $x_{11}, \ldots, s_{13}$, $x_{21}, \ldots, s_{23}$. Moreover, "K" is used as the number of subproblems, meaning "K" $= 2$ for the two described subproblems.
The master problem returns π-values and κ-values which will then be used in the subproblems. The code for the subproblems looks like the following.

Listing 3: Subproblem task 4

```
using JuMP, GLPK, LinearAlgebra

include("task4-Data.jl")

# dual variables from master problem:
# it 1:
# piVal = [0, 0, 0]
# kappa = [6.0, 4.0]
# it 2:
piVal = [0.0, -0.5, 0.0]
kappa = [6.0, 7.0]


for k in 1:K
    sub = Model(GLPK.Optimizer)

    # 1) Variables for 3 periods each
    @variable(sub, x[1:3] >= 0)
    @variable(sub, y[1:3], Bin)
    @variable(sub, s[1:3] >= 0)

```

```
    # 2) Build a single 9-vector [ x; y; s ]
    vec = vcat(x,y,s)

    # 3) Reduced-cost objective
    @objective(sub, Min,
        dot(CV[k],vec)  # production+setup+hold
      - dot(piVal, A0_V[k] * vec)  # capacity duals
      - kappa[k]                   # convexity dual
    )

    # 4) Flowbalance equalities (rows 1:3 of A_V, b_sub)
    @constraint(sub,
       A_V[k][1:3, :] * vec .== b_sub[k][1:3]
    )

    # 5) big-M <= constraints (rows 4:6 of A_V, b_sub)
    @constraint(sub,
       A_V[k][4:6, :] * vec .<= b_sub[k][4:6]
    )

    optimize!(sub)
    if termination_status(sub) == MOI.OPTIMAL
        rc, patt = objective_value(sub), value.(vec)
        println("--- Subproblem $k optimal:  reduced cost = $rc")
        println("   pattern = ", patt)
    else
        error("Subproblem $k failed with status ",
           termination_status(sub))
    end
end
```

Running the master problem and subproblem with the programs provided, gives the following iteration results:

iter	Master: π			Master: κ		Obj. value
	π_1	π_2	π_3	κ_1	κ_2	
1	0.0	0.0	0.0	6.0	4.0	10.0
2	0.0	-0.5	0.0	6.0	7.0	8.0

Table 4: Master problem progress.

Which means that the initial extreme point for product 2, is very good and does not require any further modifications, and after the reduced cost is made by introducing another extreme point for product 1, it is possible to get the optimal solution when there are no more reduced costs for either product 1 or product 2.

iter	Subproblem $i = 1$										Subproblem $i = 2$									
	x_{11}	x_{12}	x_{13}	y_{11}	y_{12}	y_{13}	s_{11}	s_{12}	s_{13}	obj	x_{21}	x_{22}	x_{23}	y_{21}	y_{22}	y_{23}	s_{21}	s_{22}	s_{23}	obj
1	3.0	4.0	5.0	1.0	1.0	1.0	0.0	0.0	0.0	3										0
2										0										0

Table 5: Subproblems for each iteration.

Showcasing the algorithm

Besides providing the code for running the Dantzig-Wolfe Decomposition in listing 2 and 3, it is also shown here how the algorithm works if it were to be solved by hand. Generally, the algorithm can be described as:

1. Initial extreme points and initial master solve
2. Solve the subproblems
3. Update and resolve the restricted master
4. Repeat step 2 and 3 until no more reduced cost are found in the subproblems

Step 1

The initial extreme point is revisited for both product 1 and product 2. These extreme points can be any set of feasible solutions.

$$X[1] = \begin{bmatrix} 7 & 0 & 5 & 1 & 0 & 1 & 4 & 0 & 0 \end{bmatrix}^\top$$
$$X[2] = \begin{bmatrix} 0 & 5 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^\top$$

Here the cost for each product is calculated by the objective function:

$$obj_{P1} = \sum_{t=1}^{3} c_1 y_{1t} + h_1 s_{1t} = 1 \cdot 2 + 1 \cdot 4 = 6$$
$$obj_{P2} = \sum_{t=1}^{3} c_2 y_{2t} + h_2 s_{2t} = 2 \cdot 2 + 1 \cdot 0 = 4$$
$$obj = obj_{P1} + obj_{P2} = 6 + 4 = 10$$

With the initial extreme points as the only columns in the master problem, the master problem forces $\lambda_{11} = 1$ and $\lambda_{21} = 1$, since there are only one extreme points for every product. Returning the duals from these constraints gives:

$$\pi = \begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix}, \qquad \kappa = \begin{bmatrix} 6.0 & 4.0 \end{bmatrix}$$

Step 2

To solve the independent subproblems, the subproblems are solved one at a time. For product 1 subproblem, plugging in the values:

$$\begin{aligned} \min_{x,y,s} \quad & \sum_{t=1}^{m} ((1 - \pi_t \cdot 2)y_{1t} + 1 \cdot s_{1t} - \pi_t x_{1t}) - \kappa_1 \\ \text{subject to} \quad & x_{11} - s_{11} = 3 \\ & s_{1t-1} + x_{1t} - s_{1t} = d_{1t}, \quad \forall t = 2, 3 \\ & x_{1t} \leq M_1 y_{1t}, \quad \forall t = 1, \ldots, m \\ & x_{1t}, s_{1t} \geq 0, \quad \forall t = 1, \ldots, m \\ & y_{1t} \in \{0, 1\} \end{aligned}$$

For the π- and κ-values generated in step 1, and minimizing for x, y, s. The results are (as seen in table 5):

$$x = [3, 4, 5] \quad y = [1, 1, 1] \quad s = [0, 0, 0]$$

The reduced cost of adding this pattern can be calculated now as inserting these variable values in the original objective and see if adding this column reduce the cost of the master.

$$\begin{aligned} obj'_{P1} &= \sum_{t=1}^{3} c_1 y_{1t} + h_1 s_{1t} = 1 \cdot 3 + 1 \cdot 0 = 3 \\ redcost_{P1} &= obj'_{P1} - obj_{P1} = 3 - 6 = -3 \end{aligned}$$

Since adding this column introduces a reduced cost, it will be added to the master problem. As seen in table 5, it is seen that the subproblem for product 2 do not impose a reduced cost and therefore it will not be calculated in this showcase.

Step 3

The column found in step 2 is now added to the master problem, meaning the extreme points looks like the following:

$$\begin{aligned} X[1] &= \begin{bmatrix} 7 & 0 & 5 & 1 & 0 & 1 & 4 & 0 & 0 \\ 3 & 4 & 5 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^\top \\ X[2] &= \begin{bmatrix} 0 & 5 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^\top \end{aligned}$$

With this additional column/extreme point, the master problem is solved again, this time will λ for product 1 have 2 values for the 2 extreme points meaning $\lambda_{11} + \lambda_{12} = 1$, and λ for product 2 will still just have 1 value, meaning $\lambda_{21} = 1$.

Solving the master problem imposing this will give the values for λ:

$$\lambda_{11} = \frac{1}{3}, \quad \lambda_{12} = \frac{2}{3}, \quad \lambda_{21} = 1$$

and the objective value can be calculated as:

$$obj''_{P1} = obj_{P1} \cdot \lambda_{11} + \sum_{t=1}^{3}(c_1 y_{1t} + h_1 s_{1t}) \cdot \lambda_{12} = 6 \cdot 1/3 + (1 \cdot 3 + 1 \cdot 0) \cdot 2/3 = 4$$

$$obj''_{P2} = \sum_{t=1}^{3}(c_2 y_{2t} + h_2 s_{2t}) = 2 \cdot 2 + 1 \cdot 0 = 4$$

$$obj'' = obj''_{P1} + obj''_{P2} = 4 + 4 = 8$$

Returning the duals from the restricted master problem gives:

$$\pi = \begin{bmatrix} 0.0 & -0.5 & 0.0 \end{bmatrix}, \qquad \kappa = \begin{bmatrix} 6.0 & 7.0 \end{bmatrix}$$

Step 4

The independent subproblems is solved again, with the new π- and κ-values, but otherwise this is the same as in step 2. As seen in the iteration table, table 5, there are no reduced costs for any of the subproblems, meaning that the algorithm is done. The final objective value for the Dantzig-Wolfe decomposition on this is:

$$z^{DW} = 8$$

Task 5

The CLSP-model given is implemented in julia for solving the IP to optimality. The parameters given is defined, and the code snippet in Listing 4 shows how the IP-model is implemented in julia.

Listing 4: IP julia implementation

```
1 using JuMP
2 using Gurobi
3      model = Model(Gurobi.Optimizer)
4 # Variables
5 @variable(model, y[1:num_products, 1:num_periods], Bin)
     # setup indicator
6 @variable(model, 0 <= x[1:num_products, 1:num_periods] <= M2,
     Int)   # production qty
7 @variable(model, 0 <= s[1:num_products, 1:num_periods] <= M3,
     Int)   # end-period inventory
8
```

```
# Objective
@objective(model, Min, sum(c[i] * y[i,t] + h[i] * s[i,t] for i
   in 1:num_products, t in 1:num_periods))

# Inventorybalance and setup constraints
# first period: x_i1 = d_i1 + s_i1
@constraint(model,[i in 1:num_products], x[i,1] == demand[i,1] +
   s[i,1])
# second period: t >= 2: s_{i,t-1} + x_{i,t} = d_{i,t} + s_{i,t}
@constraint(model,[i in 1:num_products, t in 2:num_periods],
   s[i,t-1] + x[i,t] == demand[i,t] + s[i,t])
# linking production to setup
@constraint(model,[i in 1:num_products, t in 1:num_periods],
   x[i,t] <= M1 * y[i,t])
# capacity constraint
@constraint(model,[t in 1:num_periods], sum(x[i,t] for i in
   1:num_products) + sum(q[i] * y[i,t] for i in 1:num_products)
   <= C)
# Solve
optimize!(model)
```

When solving the LP-relaxation of the model, we relax all integrality constraints on the three variables, meaning:

$$0 \leq y_{it} \leq 1, \quad 0 \leq x_{it} \leq M_2, \quad 0 \leq s_{it} \leq M_3 \quad \forall i = 1, ..., n, t = 1, ..., m$$

Other than this, the model is identical to the Listing 4, so the changed variables can be implemented as:

Listing 5: LP-variables changes in julia

```
# Variables
@variable(model, 0 <= y[1:num_products, 1:num_periods] <= 1)
          # setup indicator
@variable(model, 0 <= x[1:num_products, 1:num_periods] <= M2)
   # production units
@variable(model, 0 <= s[1:num_products, 1:num_periods] <= M3)
   # end-period inventory
```

The objective values for the two models is found to be:

$$z^{IP} = 9 \qquad z^{LP} = 6.34$$

With the following variable results:

(a) IP

P1-var	Val	P2-var	Val
x_{11}	5	x_{21}	0
x_{12}	2	x_{22}	5
x_{13}	5	x_{23}	2
y_{11}	1	y_{21}	0
y_{12}	1	y_{22}	1
y_{13}	1	y_{23}	1
s_{11}	2	s_{21}	0
s_{12}	0	s_{22}	0
s_{13}	0	s_{23}	0

(b) LP

P1-var	Val	P2-var	Val
x_{11}	4.14	x_{21}	0
x_{12}	2.86	x_{22}	5
x_{13}	5	x_{23}	2
y_{11}	0.83	y_{21}	0
y_{12}	0.57	y_{22}	1
y_{13}	1	y_{23}	0.4
s_{11}	1.14	s_{21}	0
s_{12}	0	s_{22}	0
s_{13}	0	s_{23}	0

Table 6: Variable values for the IP and LP

Objective Value Comparison

Taking the objective value from the Dantzig-Wolfe solution and comparing it to the IP and LP solution shows:

The results are expected. This is because the Dantzig-Wolfe decomposition captures much more of the problem by creating columns that are integer-feasible (by having $y \in \{0, 1\}$). This is a much more realistic result than the naive LP-relaxation approach, which can be seen by the gap from the optimum:

$$\text{LP}^{gap} \approx 30\%, \qquad \text{DW}^{gap} \approx 11\%$$

In general we expect:

$$z^{IP} \geq z^{DW} \geq z^{LP}$$

which is exactly what is found in this case.

Task 6

Since it is established that the DW solution performs better than the LP relaxation, it is time to investigate this solution further.
First, it is necessary to convert the λ-variables back to the original variables. This is done to establish an idea of where the DW solution is infeasible in the original problem.
This is done in the following way:

$$vars = X \cdot \lambda$$

where X is the final columns, consisting of $X[1]$ and $X[2]$ and all the λ consist of all the λ-variables, meaning λ_{11}, λ_{12} and λ_{21}.
This results in the following y-values: It is clear that there is one fractional y-variable, that

y	Value
y_{11}	1
y_{12}	2/3
y_{13}	1
y_{21}	0
y_{22}	1
y_{23}	1

Table 7: y-values after DW decomposition

is y_{12}. To investigate the solution further, branching will be done on y_{12} fixing the value of y_{12} to 1 and 0.
The branching and evaluation is done in the same way as in task 4, with the change that in one iteration of the DW decomposition $y_{12} = 0$ and in a second iteration of the DW decompotistion, $y_{12} = 1$.
This is done in the following way when defining the data in julia:

Listing 6: Branching on y_{12}

```
1 b = [3, 0, 4, 5, 5, 2, 0,0,0,0,0,0, 1,0, 10,10,10] # change
     fixed branch value to 1
2 # b = [3, 0, 4, 5, 5, 2, 0,0,0,0,0,0, 0,0, 10,10,10] # change
     fixed branch value to 0
```

This will cause the subproblem to enforce the value branched on by introducing the following constraint in the subproblem script:

Listing 7: Constrained subproblem for y_{12}

```
1     # 6) Branch constraint
2     @constraint(sub,
3        dot(A_V[k][7,:],vec) == b_sub[k][7]
4     )
```

With the implementation of this and running the DW Decomposition in the same manual way as in Task 4, we receive the following results - following the same setup as in task 4, but this time running the DW Decomposition twice with the fixed values.

Fixed $y_{12} = 0$						
iter	**Master:** π			**Master:** κ		**Obj. value**
	π_1	π_2	π_3	κ_1	κ_2	
1	0.0	0.0	0.0	6.0	4.0	10.0

Table 8: Master problem progress for $y_{12} = 0$

Fixed $y_{12} = 0$																				
iter	**Subproblem** $i = 1$										**Subproblem** $i = 2$									
	x_{11}	x_{12}	x_{13}	y_{11}	y_{12}	y_{13}	s_{11}	s_{12}	s_{13}	obj	x_{21}	x_{22}	x_{23}	y_{21}	y_{22}	y_{23}	s_{21}	s_{22}	s_{23}	obj
1										0										0

Table 9: Subproblems for fixed $y_{12} = 0$

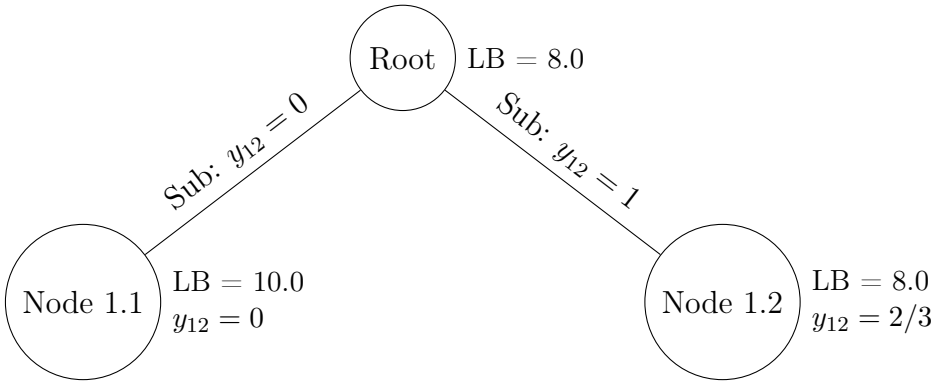
Fixed $y_{12} = 1$						
iter	**Master:** π			**Master:** κ		**Obj. value**
	π_1	π_2	π_3	κ_1	κ_2	
1	0.0	0.0	0.0	6.0	4.0	10.0
2	0.0	-0.5	0.0	6.0	7.0	8.0

Table 10: Master problem progress for $y_{12} = 1$

Fixed $y_{12} = 1$																				
iter	**Subproblem** $i = 1$										**Subproblem** $i = 2$									
	x_{11}	x_{12}	x_{13}	y_{11}	y_{12}	y_{13}	s_{11}	s_{12}	s_{13}	obj	x_{21}	x_{22}	x_{23}	y_{21}	y_{22}	y_{23}	s_{21}	s_{22}	s_{23}	obj
1	3.0	4.0	5.0	1.0	1.0	1.0	0.0	0.0	0.0	3										0
2										0										0

Table 11: Subproblems for fixed $y_{12} = 1$

All of these objective values can be summarized in the following branch and bound tree:



From the branch-and-bound tree, we conclude that the relaxation of the master problem gives a lower bound of 8.0. Branching on the fractional variable y_{12} produces one branch $y_{12} = 0$ with a feasible integer solution of value 10.0, and another branch $y_{12} = 1$ that still results in a fractional solution with lower bound 8.0. This indicates that the optimal integer solution might lie in the unexplored subtree under $y_{12} = 1$, but in Node 1.2, where the subproblem was constrained to generate only patterns with $y_{12} = 1$ the reconstructed value of y_{12} in the master problem remains fractional.
This happens because the λ-variables forming the convex combination of columns can still be fractional, and DW decomposition does not impose integrality on the original variables directly. For now, the branching is done, but if further branching were to be done, it would be done on the same variable as common in branch-and-price algorithms.

Task 7

For the decomposition from task 3 to satisfy to have identical sub-problem, we observe the two independent sub-problem for the two products. These two sub-problems (equation 13-18) and (equation 19-24) must satisfy the following assumptions.

Identical sub-problem assumptions

$$\forall k, l \in K: \quad A^k_{V_k} = A^l_{V_l} \tag{25}$$
$$\forall k, l \in K: \quad c_{V_k} = c_{V_l} \tag{26}$$
$$\forall k, l \in K: \quad A^0_{V_k} = A^0_{V_k} \tag{27}$$
$$\forall k, l \in K: \quad b^k = b^l \tag{28}$$

From this we see that our sub-problems does not satisfy the assumptions since the objective (13) is not equal to (19), meaning that assumption (26) and (27) is not met and therefore we cannot apply identical sub-problems for our decomposition.

Task 8

Transition from Model [DW] to Model [CLSP]

To get from model [DW] to [CLSP], we apply the Dantzig-Wolfe decomposition. Model [CLSP] is a specialized, simplified version of the general Dantzig-Wolfe master problem [DW], tailored to a cost-minimizing production planning problem.
We start with the general Dantzig-Wolfe master problem:

$$\text{[DW]} \quad \min \sum_{k \in K} c^k \bar{V}^k \lambda_k$$
$$\text{s.t.} \quad A_0 \sum_{k \in K} \bar{V}^k \lambda_k \leq b_0$$
$$\sum_{k \in K} \lambda_k = 1, \quad \lambda_k \geq 0 \quad \forall k \in K$$

This is a general model where each λ_k represents a convex combination of feasible solutions (columns) generated by solving subproblems.

To derive the model for the multi-item capacitated lot-sizing problem with setup times, we observe that the problem decomposes by item. That is, for each item $i = 1, \ldots, n$, we solve an independent subproblem generating a set of feasible production plans Ω_i.
Each plan $p \in \Omega_i$ represents an extreme point (column), and we define:

- λ^i_p: the selection of production plan p for item i,
- γ^i_p: the total cost of production plan p for item i,

- α_{itp}: the amount of capacity used in time t by plan p for item i.

Thus, the master problem becomes:

$$[\text{CLSP}] \quad \min 0 \sum_{i=1}^{n} \sum_{p \in \Omega_i} \gamma_{ip} \lambda_p^i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{p \in \Omega_i} \alpha_{itp} \lambda_p^i \leq C \quad \forall t = 1, \ldots, m$$

$$\sum_{p \in \Omega_i} \lambda_p^i = 1 \quad \forall i = 1, \ldots, n$$

$$\lambda_p^i \geq 0 \quad \forall i = 1, \ldots, n, \ \forall p \in \Omega_i$$

Pricing Problem in Column Generation

In the column generation approach used to solve the CLSP model, the pricing problem is a subproblem that identifies whether there are new columns that could improve the current solution to the master problem.
For item i, and given the current dual prices π_t from the master problem , we want to find a production plan p that minimizes the reduced cost. If the reduced cost is negative, the new plan p is added to the master problem as a new variable λ_p^i, and if no such plan exists for any item, the current solution is optimal.

Pros and cons of implementing model [CLSP] instead of model [DW]

The CLSP model is more simple and therefore easier to understand and implement. Moreover, it is also more efficient since column generation can efficiently handle large-scale problems by focusing on promising columns and thereby reduce computational complexity. However, the con is that column generation also requires solving multiple subproblems iteratively, which can be computationally intensive. Another con is that CLPS model requires careful implementation and tuning to ensure convergence and optimality. Lastly, the simplified model may lose some generality and flexibility compared to the original Dantzig-Wolfe model.

Task 9

Since λ_i^p is a binary variable thats selects plan p, we are able to get a vail model by enforcing λ_i^p to be integer as long as each production plan $p \in \Omega_i$ represents a complete and feasible production plan for item i across all time periods. The set Ω_i includes **all** feasible production plans for item i. That is, the model includes a full representation of the feasible solution space. The constraint (29) ensures that exactly one plan is selected per item when $\lambda_p^i \in$

$\{0, 1\}$, and therefore we can get a valid model when λ_p^i is integer.

$$\sum_{p \in \Omega_i} \lambda_p^i = 1 \quad \forall i \tag{29}$$

However, the number of production plans $|\Omega_i|$ is typically exponential in the number of time periods. Therefore, solving the full integer version of [CLSP] is computationally intractable in practice.

Task 10

```
function f(x)
    return x % 5 + 1
end

x = 19640+ 203947
f(x)

julia> 3
```

What is the paper about? The paper is a comprehensive survey on shortest path problems with resource constraints (SPPRC), a key subproblem in vehicle routing and crew scheduling solved via column generation. It provides a generic formulation, classifies different SPPRC variants, discusses complex modeling issues, and reviews solution methods like dynamic programming. The paper focuses on modeling, solving, and understanding the complexity and variations of SPPRC.
How is the paper related to Dantzig-Wolfe decomposition? The SPPRC often appears as the subproblem in a Dantzig-Wolfe decomposition approach for vehicle routing and crew scheduling. In such setups, the master problem is a set partitioning model, and SPPRC generates columns (routes or schedules) with resource constraints. Thus, SPPRC's efficient solution is critical to the success of Dantzig-Wolfe-based column generation methods.
What is the contribution of the paper? The paper's main contributions are: (1) proposing a classification and generic formulation for all SPPRC variants, (2) addressing non-trivial modeling issues for resources and constraints, and (3) surveying the literature, highlighting important methodological innovations and applications. It serves as a foundational reference for anyone tackling resource-constrained routing or scheduling problems.

The answers are useful because they summarize the main points of the paper in a concise way. They correctly link the paper's focus to Dantzig-Wolfe decomposition and highlight the contributions.
One point of critique: The answers could be improved by being more specific about the paper's broader impact, such as emphasizing that it not only surveys the field but also

standardizes terminology and formalism for SPPRC. Right now, the summary captures what the paper does but not fully why that is significant.
The description of the contribution mentions 'solution methods' but does not name dynamic programming, Lagrangian relaxation, or constraint programming explicitly - methods that are key parts of the paper's discussion.

Contribution

	Karen s196140	Aksel s203947
Task 1	50%	50%
Task 2	50%	50%
Task 3	20%	80%
Task 4	40%	60%
Task 5	20%	80%
Task 6	60%	40%
Task 7	50%	50%
Task 8	80%	20%
Task 9	80%	20%
Task 10	50%	50%

Appendix - Scripts

Attached to this assignment is the following files. The crucial parts of the scripts are already included in the respective task, but if they are to be run individually, this is the structure:

1. **Data definitions for task 4:** task4-Data.jl
2. **Master problem iteration 1 for task 4:** task4-MasterIt1.jl
3. **Master problem iteration 2 for task 4:** task4-MasterIt2.jl
4. **Subproblem iterations for task 4:** task4-sub.jl
5. **IP/LP solve of the original CLSP-model:** Task5.jl
6. **Variable transformation for results task 4:** BranchPrice_4.jl
7. **Data definitions for task 6:** task6-Data.jl
8. **Master problem iteration 1 for task 6:** task6-MasterIt1.jl
9. **Master problem iteration 2 for task 6:** task6-MasterIt2.jl
10. **Subproblem iterations for task 6:** task6-sub.jl
11. **Variable transformation for results task 6:** BranchPrice_6.jl