42136 - Assignment 1

Spring 22, 42136
Assignment 1
March 18, 2022

Carl Frederik Grønvald, s184482
Anton Ruby Larsen, s174356

Danmarks
Tekniske Universitet

1 | A

In this assignment we will work with the decomposition technique called Danzig-Wolfe decomposition. To illustrate how this technique works we will apply it to the Multiple Knapsack Problem with Setup (MKPS) given in 1.0.1.

$$\max \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_t} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it} \tag{1.0.1a}$$

s.t.

$$\sum_{i=1}^{N}\sum_{j=1}^{n_t} a_{ij}x_{ijt} + \sum_{i=1}^{N} d_i y_{it} \leq b_t \quad \forall t \in \{1,\dots,T\} \tag{1.0.1b}$$

$$x_{ijt} \leq y_{it} \quad \forall i \in \{1,\dots,N\}, j \in \{1,\dots,n_i\}, t \in \{1,\dots,T\} \tag{1.0.1c}$$

$$\sum_{t=1}^{T} y_{it} \leq 1 \quad \forall i \in \{1,\dots,N\} \tag{1.0.1d}$$

$$x_{ijt} \in \{0,1\} \forall i \in \{1,\dots,N\}, j \in \{1,\dots,n_i\}, t \in \{1,\dots,T\} \tag{1.0.1e}$$

$$y_{it} \in \{0,1\} \forall i \in \{1,\dots,N\}, t \in \{1,\dots,T\} \tag{1.0.1f}$$

To understand the problem we will first explain the objective function. Here we first have a portfolio of products x_{ijt}. Each product j comes from a class i and can be produced at a production facility t. These products all have a profit c_{ijt} which is specific to the product, class and production facility. If some production facility t, chooses to produce a class i, then it has a setup cost f_{it}. Hence the objective function is as follows.

$$\underbrace{\sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_t} c_{ijt}x_{ijt}}_{income} + \underbrace{\sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it}}_{setup\ cost} = \text{profit}$$

Next we have the constraints. First each production facility must hold a budget b_t. The expenses consist of two parts. First we have the product specific expenses $a_{ij}x_{ijt}$ which could be raw material and man power at the production lines. Secondly each production facility also has some class dependent expenses $d_i y_{it}$ which could be administration of this type of products which does not scale with the production within the class. Hence we have

$$\underbrace{\sum_{i=1}^{N}\sum_{j=1}^{n_t} a_{ij}x_{ijt}}_{product\ expenses} + \underbrace{\sum_{i=1}^{N} d_i y_{it}}_{class\ expenses} \leq \underbrace{b_t}_{facility\ budget}$$

Next we must make sure our problem formulation makes sure only production facilities that have been setup to produce class i produces class i. This is enforced by constraint 1.0.1c.

The last affine constraint is 1.0.1d. This constraint makes sure that a class of products is at most produced by one production facility.

Lastly we have constraint 1.0.1e and 1.0.1f which says x_{ijt} and y_{it} must be binary. The reason is that they work as indicators throughout the problem. Does production facility t produce product j within class i? If yes, then $x_{ijt} = 1$ and if not then $x_{ijt} = 0$. The other question we must be able to answer is if a production facility is capable of producing product class i. This is what y_{it} answers.

2 | B

We are now given 20 instances of the MKPS, 1.0.1. Together with the instances we are provided a Julia program `readMKPS()` which takes an instance and then output all the relevant vectors and matrices. We are now to solve the 20 instances and their LP relaxations but because some of the instances are really hard we have enforced a time limit of 20 minutes per instance. We have chosen the CPLEX solver to solve the MIPs generated from the instances. In table 2.1 we see the 10 first instances for which $T = 5$ and $N = 10$. From the table we see that each instance is solved to optimality and the time spent is far from reaching the time limit.

	1	2	3	4	5	6	7	8	9	10
Objective Value	36278.0	51161.0	104146.0	83876.0	79538.0	122764.0	80202.0	88204.0	125670.0	122752.0
Upper Bound	36278.0	51161.0	104146.0	83876.0	79538.0	122764.0	80202.0	88204.0	125670.0	122752.0
Relative Gap	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Time Spent [s]	0.804	1.465	3.878	13.979	1.423	5.765	2.458	9.927	4.085	8.281
LP Relaxation	62788.7	68364.5	128100.2	101892.5	103006.2	129864.8	98963.5	102170.5	133341.6	136218.9

Table 2.1 – Instances with $T = 5$ and $N = 10$.

Next we solve the 10 instances for which $T = 10$ and $N = 30$. These problems are much larger than the previous 10. The problems in table 2.1 had on average around 3500 binary variables and 3500 affine constraints where problems in table 2.2 has around 15000 binary variables and 15000 affine constraints. Hence due to the combinatorial nature of integer programs the latter problems will be much harder to solve which we also see in table 2.2. We see all problems except one use all 20 minutes without reaching optimality.

	1	2	3	4	5	6	7	8	9	10
Objective Value	814428.0	588095.0	897384.0	603221.0	681404.0	670827.0	594031.0	896416.0	792979.0	620145.0
Upper Bound	814432.8	588095.0	897386.6	603259.1	681441.5	670854.2	594066.8	896464.0	793015.5	620173.1
Relative Gap	5.894e-6	0.0	2.868e-6	6.322e-5	5.501e-5	4.050e-5	6.035e-5	5.353e-5	4.602e-5	4.532e-5
Time Spent [s]	1200.1	347.0	1200.1	1201.1	1200.2	1200.3	1200.1	1203.0	1200.5	1203.3
LP Relaxation	1.1403e6	1.0063e6	1.2202e6	1.0276e6	1.1183e6	1.1464e6	1.0264e6	1.2208e6	1.1482e6	1.0971e6

Table 2.2 – Instances with $T = 10$ and $N = 30$.

3 | C

Danzig Wolfe reformulation is a method for rewriting a linear or integer program. Instead of solving a single program, the constraints of the program is split into a part which we convexify by use of Minkowski-Weyl's theorem and a remaning part for which we do a substitution of variables due to the convexification. If the program has a special structure we can use the Danzig Wolfe reformulation in combination with column generation and obtain a significant speed up. [1] writes this especially holds for exponentially large problems.
We will now explore the structure of the given instance of the Multiple Knapsack Problem with Setup (MKPS), "mini-instance.txt", to see if it has some of this wanted structure. It consists of two knapsacks, two classes and two items per class. Hence $T = 2$, $N = 2$ and $n_i = 2 \forall i \in \{1, 2\}$. The constraint matrix of the problem is given in the table 3.1. Note that constraints 5 and 6 are not written in the matrix, since it would take up too much space. They require that $x_{ijt} \in \{0, 1\}$ for $i \in \{1, 2\}$, $j \in \{1, 2\}$, $t \in \{1, 2\}$, and $y_{it} \in \{0, 1\}$ for $i \in \{1, 2\}$, $t \in \{1, 2\}$.

row variable	x_{111}	x_{112}	x_{121}	x_{122}	x_{211}	x_{212}	x_{221}	x_{222}	y_{11}	y_{12}	y_{21}	y_{22}	RHS	constraint no
1	1		4		4		8		8		9		20	2 or 1.0.1b
2		1		4		4		8		8		9	25	2 or 1.0.1b
3	1								-1				0	3 or 1.0.1c
4		1								-1			0	3 or 1.0.1c
5			1						-1				0	3 or 1.0.1c
6				1						-1			0	3 or 1.0.1c
7									1	1			1	4 or 1.0.1d
8					1						-1		0	3 or 1.0.1c
9						1						-1	0	3 or 1.0.1c
10							1				-1		0	3 or 1.0.1c
11								1				-1	0	3 or 1.0.1c
12											1	1	1	4 or 1.0.1d

Table 3.1 – Constraint matrix for the instance of 1.0.1 given in the file "mini-instance.txt". The column *constraint no* refers to numbering in the assignment description.

4 | D

In this exercise we are to split the constraint matrix 3.1 up into a constraint block which belongs to the master and T independent blocks each belonging to a sub-problem. This structure is found on slide 26, week 3 and here given in equation 4.0.1.

$$A = \begin{bmatrix} A^0_{V^1} & A^0_{V^2} & \cdots & A^0_{V^T} \\ A^1_{V^1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & A^2_{V^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & A^T_{V^T} \end{bmatrix} \tag{4.0.1}$$

We are told that constraint (4), or in this report 1.0.1d, must belong to the master problem and hence constraint 1.0.1b and 1.0.1c split up into 2 independent sets/blocks. Splitting on the index t gives us these independent blocks as shown in table 4.1.

row variable	x_{111}	x_{121}	x_{211}	x_{221}	y_{11}	y_{21}	x_{112}	x_{122}	x_{212}	x_{222}	y_{12}	y_{22}	RHS	constraint no
7					1						1		1	4 or 1.0.1d
12						1						1	1	4 or 1.0.1d
1	1	4	4	8	8	9							20	2 or 1.0.1b
3	1				-1								0	3 or 1.0.1c
5		1			-1								0	3 or 1.0.1c
8			1			-1							0	3 or 1.0.1c
10				1		-1							0	3 or 1.0.1c
2							1	4	4	8	8	9	25	2 or 1.0.1b
4							1				-1		0	3 or 1.0.1c
6								1			-1		0	3 or 1.0.1c
9									1			-1	0	3 or 1.0.1c
11										1		-1	0	3 or 1.0.1c

Table 4.1 – Here we split on the knapsacks. Yellow indicates that these rows goes into the master problem while green and blue are two blocks.

Another way one could split the 1.0.1 into master and sub-problems would be putting constraint 1.0.1b into the master and then split constraints 1.0.1c and 1.0.1d on the classes. This would give N sub-problems instead of T and the structure is shown in table 4.2. This is a much worse reformulation than the splitting on the one given in 4.1 but this we will return to.

row variable	x_{111}	x_{112}	x_{121}	x_{122}	y_{11}	y_{12}	x_{211}	x_{212}	x_{221}	x_{222}	y_{21}	y_{22}	RHS	constraint no
1	1		4		8		4		8		9		20	2 or 1.0.1b
2		1		4		8		4		8		9	25	2 or 1.0.1b
3	1				-1								0	3 or 1.0.1c
4		1				-1							0	3 or 1.0.1c
5			1		-1								0	3 or 1.0.1c
6				1		-1							0	3 or 1.0.1c
7					1	1							1	4 or 1.0.1d
8							1				-1		0	3 or 1.0.1c
9								1				-1	0	3 or 1.0.1c
10									1		-1		0	3 or 1.0.1c
11										1		-1	0	3 or 1.0.1c
12											1	1	1	4 or 1.0.1d

Table 4.2 – Here we split on the classes. Yellow indicates that these rows goes into the master problem while green and blue are two blocks.

5 | E

The master and subproblems are solved by solvers, but data is manually transferred between the two. Iterations of the solver are in table 5.1. We start out with a dummy variable λ_0 that fulfills all constraints and has coefficient -1000 in the objective function. We see that we check both subproblems every iteration for new points, and add those to the master problem. We keep track of where each extreme point in the master problem originates, so we can use the correct constraints on each extreme point.

5.1 Play-by-play analysis

First, we solve the master problem with our dummy variable, λ_0, to find the duals. We get a very large, negative κ_1 value, and a κ_2 of zero. π is just the zero vector, $\mathbf{0}$. This leads to subproblem 1 giving us a new extreme point with a positive reduced cost, which means we take it, since this is a maximization problem. Subproblem 2 has a reduced cost of zero, so we do not get a new extreme point from it.
Then, we solve our master problem again, this time with our new extreme point as the only extreme point. We still get $\pi = \mathbf{0}$, but κ has changed. Now κ_2 is very negative. Now when we solve the subproblems, we get reduced cost 0 from subproblem 1, giving us no new extreme point, but we get a positive reduced cost from subproblem 2, adding the extreme point $\mathbf{0}$.
We solve the master problem again with our two extreme points, keeping track of which subproblem they originate from. We get $\pi = \mathbf{0}$, and $\kappa = \begin{bmatrix}1 & 0\end{bmatrix}$. We solve the subproblems again with these new duals, but both give us a reduced cost of zero. Then, since no subproblems have reduced cost greater than zero, we are done. The solution is given by the last time we solved the master problem. We get $\lambda = \begin{bmatrix}1 & 1\end{bmatrix}$, which means we'll use our first extreme point for subproblem 1, and our second extreme point for subproblem 2. The extreme point for subproblem 2 is just $\mathbf{0}$, so the second knapsack is left empty. The first knapsack has both items of class one put into it. The solution has a final objective value of 1.

Problem	Input	Output
Master problem	No extreme points	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa = \begin{bmatrix} -1000 & 0 \end{bmatrix}$
Subproblem 1	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa_1 = -1000$	Reduced cost = 1001 New extreme point = $\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$
Subproblem 2	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa_2 = 0$	Reduced cost = 0 No new extreme point
Master problem	$\bar{X} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$ Originating subproblem = $\begin{bmatrix} 1 \end{bmatrix}$	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa = \begin{bmatrix} 1 & -1001 \end{bmatrix}$
Subproblem 1	$\pi = \begin{bmatrix} 0... & 0 \end{bmatrix}$ $\kappa_1 = 1$	Reduced cost = 0 No new extreme point
Subproblem 2	$\pi = \begin{bmatrix} 0... & 0 \end{bmatrix}$ $\kappa_2 = -1001$	Reduced cost = 1001 New extreme point = $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
Master problem	$\bar{X} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ Originating subproblem = $\begin{bmatrix} 1 & 2 \end{bmatrix}$	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa = \begin{bmatrix} 1 & 0 \end{bmatrix}$
Subproblem 1	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa_1 = 1$	Reduced cost = 0 No new extreme point
Subproblem 2	$\pi = \begin{bmatrix} 0 & 0 \end{bmatrix}$ $\kappa_2 = 0$	Reduced cost = 0 No new extreme point
Results	$\lambda = \begin{bmatrix} 1 & 1 \end{bmatrix}$ $x_{111} = 1 \quad x_{121} = 1 \quad y_{11} = 1$ Objective value = 1	

Table 5.1 – Iteration table solving the Danzig-Wolfe relaxation of mini-instance.txt using two subproblems with constraint 4 in the master problem and all other constraints convexified.

6 | F

The instances from figure 1 in the given assignment outline, are solved in an automated way. The instances from "instances/50/" are solved in table 6.1, while the instances from "instances/NC/" are solved in table 6.2. Every instance was limited to a 20 minute runtime.
Solving the larger instances from "instances/50/", we found that column generation found the integer solution nine out of ten times, and the one time no integer solution was found, the objective value of the Danzig-Wolfe relaxation was very close to the optimal integer solution found without column generation by the CPLEX solver. Column generation was also significantly faster than the CPLEX solver without column generation. Without column generation the CPLEX solver only ran to integer optimality in less than twenty minutes for one problem, where with column generation all instances was solved to integer optimality in an average two and a half minutes. In instance three, however, the column generation method would have needed to branch to obtain integer optimality. All in all, the Danzig Wolfe reformulation with constraint four in the master problem was a very successful way to solve these optimization problems.
In solving the smaller instances from "instances/NC/", we also wanted to investigate the importance of choosing the right constraints to convexify. It is clear that keeping constraint 4 in the master (the method named "knapsack") leads to significantly better results. We see that the knapsack split hit an integer solution without branching, nine out of ten times, and the one time it did not, it still provided a bound less than 1% from the true objective value. On the other hand, the class split, given in table 4.2, never hit an integer solution, and it provided a much worse bound. For the first instance, the bound for the class split was 73% higher than the integer optimal objective value. We see, then, that the choice of which constraint to convexify is very important for the performance of the Danzig-Wolfe decomposition.
In general for the smaller instances, Danzig-Wolfe reformulation was not an improvement over the CPLEX MIP solver. This is likely due to several reasons:

1. The CPLEX solver, once running, is incredibly fast. It is a battle-tested and highly optimized tool, written in C. In comparison, our code likely has a lot of room for optimization.

2. Column generation is most useful when a problem has a lot of variables. Since these problems are so small, the improved scaling of the column generation method has yet to kick in, and constant factors on the speed of the algorithm dominate. TODO: REFERENCE

When solving small integer problems, solving directly using a highly-optimized state of the art MIP solver is then better than implementing Danzig Wolfe decomposition.

		1	2	3	4	5
CPLEX	Objective Value	814428.0	588095.0	897384.0	603221.0	681405.0
	Upper Bound	814432.8	588095.0	897386.6	603259.1	681441.1
	Relative Gap	5.913e-6	0.0	2.858e-6	6.322e-5	5.323e-5
	Time Spent [s]	1200.2	344.2	1200.1	1201.3	1200.2
	LP Relaxation	1140184.6	1006306.2	1220186.8	1027631.6	1118250.5
Col Gen	Objective Value	814428.0	588095.0	897385.1	603221.0	681405.0
	Time Spent [s]	117.9	91.4	177.8	124.1	157.3
	Integer Solution	yes	yes	no	yes	yes
		6	7	8	9	10
CPLEX	Objective Value	670827.0	594031.0	896416.0	792979.0	620145.0
	Upper Bound	670854.2	594066.6	896464.1	793014.4	620173.4
	Relative Gap	4.052e-5	5.992e-5	5.3642e-5	4.464e-5	4.584e-5
	Time Spent[s]	1200.3	1200.2	1201.3	1202.5	1200.4
	LP Relaxation	1146391.3	1026434.3	1220757.2	1148200.9	1097063.2
Col Gen	Objective Value	670827.0	594031.0	896416.0	792979.0	620145.0
	Time Spent [s]	136.4	120.4	118.5	144.3	263.7
	Integer Solution	yes	yes	yes	yes	yes

Table 6.1 – The instances from "instances/50/" solved in an automated way. CPLEX solves an MIP problem, column generation solves a problem with ten subproblems where constraint 4 is in the master problem, and all other constraints are convexified. All instances were limited to 20 minutes of run time, and column generation is timed from first solving the master problem to the last time a subproblem finishes solving. The Integer Solution row indicates whether the Danzig Wolfe relaxation solution is integer. If it is integer, it is the optimal solution, and no further branching is necessary. If it is not, further branching is needed to find the optimal solution.

		1	2	3	4	5
CPLEX	Objective Value	36278.0	51161.0	104146.0	83876.0	79538.0
	Upper Bound	36278.0	51161.0	104146.0	83876.0	79538.0
	Relative Gap	0.0	0.0	0.0	0.0	0.0
	Time Spent [s]	1.343	4.157	7.988	25.578	3.680
	LP Relaxation	62788.7	68364.5	128100.2	101892.5	103006.2
Knapsack	Objective Value	36278.0	51161.0	104146.0	83876.0	79538.0
	Time Spent [s]	3.520	3.413	16.102	9.598	9.268
	Integer Solution	yes	yes	yes	yes	yes
Class	Objective Value	62788.7	68364.5	128100.2	101892.5	103006.2
	Time Spent [s]	2.414	2.508	2.418	1.200	2.519
	Integer Solution	no	no	no	no	no
		6	7	8	9	10
CPLEX	Objective Value	122764.0	80202.0	88204.0	125670.0	122752.0
	Upper Bound	122764.0	80202.0	88204.0	125670.0	122752.0
	Relative Gap	0.0	0.0	0.0	0.0	0.0
	Time Spent[s]	4.957	2.261	9.059	3.786	7.278
	LP Relaxation	129864.8	98963.5	102170.5	133341.6	136218.9
Knapsack	Objective Value	122764.0	80202.0	88204.0	126034.0	122752.0
	Time Spent[s]	22.311	3.137	4.278	5.812	8.685
	Integer Solution	yes	yes	yes	no	yes
Class	Objective Value	129864.8	98963.5	102170.5	133341.6	136218.9
	Time Spent [s]	2.297	2.456	3.398	2.203	1.799
	Integer Solution	no	no	no	no	no

Table 6.2 – The instances from "instances/NC/" in the project description figure 1 solved in an automated way. CPLEX solves an MIP, and column generation is solved in two different ways. One is with constraint 4 in the master and the rest of the constraints convexified, named "knapsack", another with constraint 2 in the master and the rest of the constraints convexified, named "class". Both are solved using subproblems. The knapsack column generation has five subproblems, and the class column generation has ten. The Integer Solution row indicates whether the DW relaxation solution is integer. If the solution is not integer, further branching is needed to find the optimal solution.

6.1 Automation of the column generation procedure

To automate the column generation we have used the provided file, "ColGenGeneric.jl", as a starting point. from figure 6.1 we see the manual procedure on the right and the automated procedure on the left. The manual procedure shows in the red box the first solve of the master problem. The optimal dual variables are then passed to the first out of two subproblems which is marked by the green square.

On the left we then see the corresponding pieces in the automated code. In the first place everything is put into a while loop which runs until all subproblems of the iteration produces non positive values. Next up is the way we solve the master problem, which is done in exactly the same way as the manual procedure. The subproblems are controlled by a for loop iterating over the k blocks. The update of the master problem if a sub problem results in a positive reduced cost is put into a function marked by the blue square.

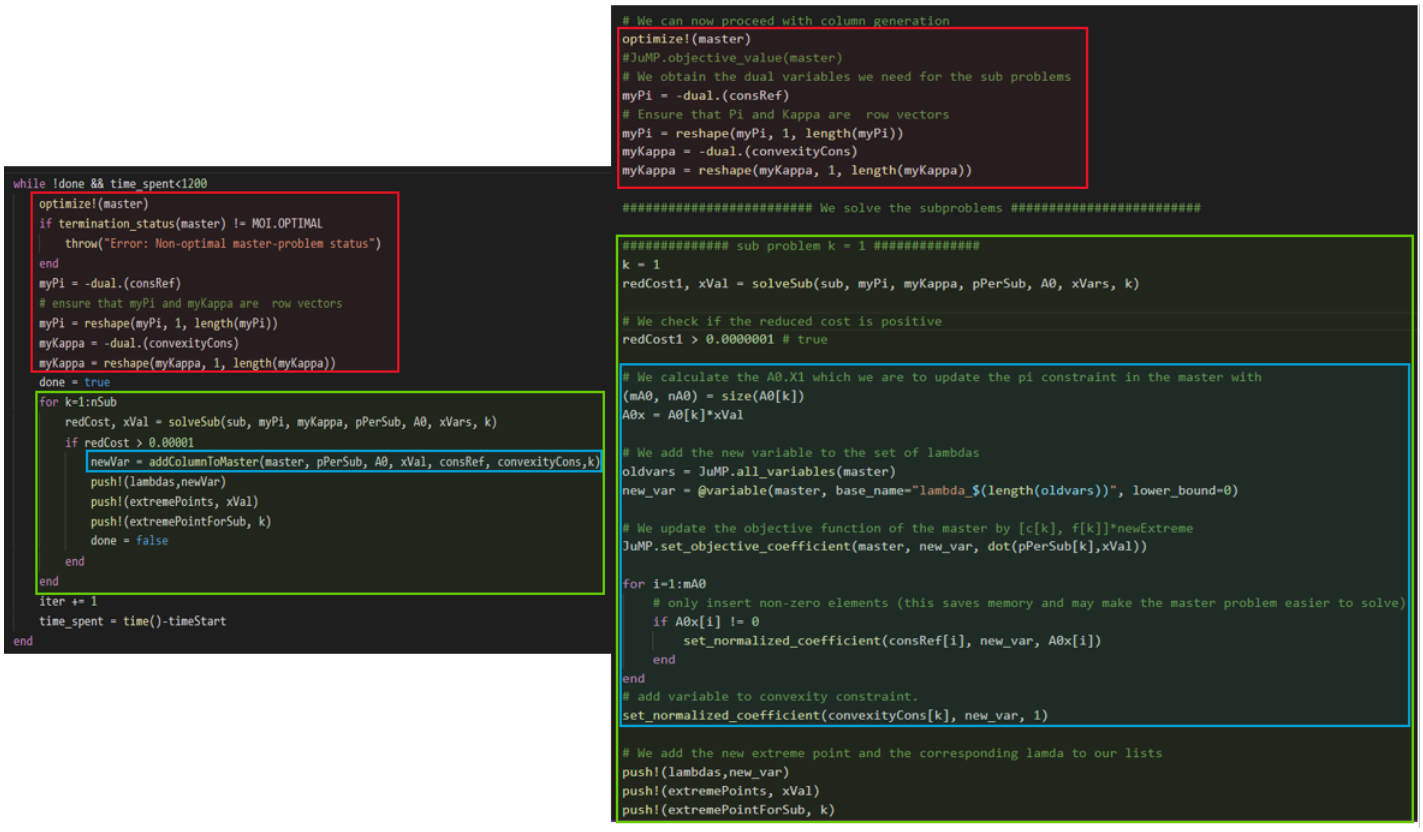

Figure 6.1 – Code changes in autoomating the column generation

7 | G

We are given another instance of the MKPS, mini-instance2. We will inspect it, and find that when keeping constraint four in the master problem and convexifying the rest, it is actually an example of identical subproblems, which allows some useful simplifications when solving the problem. The coefficients for the constraints for mini-instance2 are seen in the table below. All constraints are less than or equal constraints.

x_{111}	x_{112}	x_{121}	x_{122}	x_{211}	x_{212}	x_{221}	x_{222}	y_{11}	y_{21}	y_{12}	y_{22}	RHS	constraint no
1		4		4		8		8	9			30	2
	1		4		4		8			8	9	30	2
1								-1				0	3
	1									-1		0	3
		1						-1				0	3
			1							-1		0	3
				1					-1			0	3
					1						-1	0	3
						1			-1			0	3
							1				-1	0	3
								1		1		1	4
									1		1	1	4

(7.0.1)

By keeping constraint 4 in the master and splitting along knapsacks, we can construct the two following subproblems:

x_{111}	x_{121}	x_{211}	x_{221}	y_{11}	y_{21}	RHS	constraint no
1	4	4	8	8	9	30	2
1				-1		0	3
	1			-1		0	3
		1			-1	0	3
			1		-1	0	3

(7.0.2)

and

x_{112}	x_{122}	x_{212}	x_{222}	y_{12}	y_{22}	RHS	constraint no
1	4	4	8	8	9	30	2
1				-1		0	3
	1			-1		0	3
		1			-1	0	3
			1		-1	0	3

(7.0.3)

We see that the two subproblems have exactly identical structure; that is, their coefficient matrices $A^1_{V^1}$ and $A^2_{V^2}$ are identical, as well as the right-hand sides. This is one of the requirements for the identical subproblem assumption.
The master problem will then be

y_{11}	y_{21}	y_{12}	y_{22}	RHS	constraint no
1		1		1	4
	1		1	1	4

(7.0.4)

The split between y_{21} and y_{12} indicates the split into subproblems. The two sides represent the coefficient matrices of the subproblems in the master problem, $A^0_{V^1}$ and $A^0_{V^2}$, and they are identical, which is another requirement for the identical subproblem assumption.
Lastly, we will look at the objective function

x_{111}	x_{121}	x_{211}	x_{221}	y_{11}	y_{21}	x_{112}	x_{122}	x_{212}	x_{222}	y_{12}	y_{22}
5	8	10	3	-12	-11	5	8	10	3	-12	-11

(7.0.5)

Again, the split down the middle is the split between subproblem 1 and subproblem 2. We see that the coefficients in the objective function are identical for both problems, $c_{V^1} = c_{V^2}$, and this is the last requirement for the identical subproblem assumption. We have then fulfilled all three, and the decomposition satisfies the identical subproblem assumption when splitting along knapsacks. The difference between this problem and the previously examined mini-instance is that the two knapsacks in this problem have the same capacity/budget, represented by b_t in constraint 2.

8 | H

When we have a decomposition satisfying the identical subproblem assumption, we can make some simplifications. When solving the subproblem, we only need to solve a single subproblem, since the extreme point from this one subproblem will be the same as for all the other subproblems. Instead of having a lambda vector per subproblem, we now just need a single lambda vector, where $\mathbf{1}\lambda \leq |K|$, since we only have a single set of extreme points.
The extreme point $\mathbf{0}$ represents two empty knapsacks, and is a feasible solution. This allows a further simplification of the convexity constraint to $\mathbf{1}\lambda \leq 2$, and we then don't need to include the point $\mathbf{0}$ in our extreme point matrix $\bar{X}_1$.
Then, we get the following master problem:

$$max \begin{bmatrix} 5 & 8 & 10 & 3 & -12 & -11 \end{bmatrix} \bar{X}_1 \lambda \tag{8.0.1a}$$

$$s.t.$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \bar{X}_1 \lambda \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{8.0.1b}$$

$$\mathbf{1}\lambda \leq 2 \tag{8.0.1c}$$

$$\lambda \geq 0 \tag{8.0.1d}$$

And the following subproblem:

$$max\, \bar{c}^* = \begin{bmatrix} 5 & 8 & 10 & 3 & -12 & -11 \end{bmatrix} x - \pi \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x \tag{8.0.2a}$$

$$s.t.$$

$$\begin{bmatrix} 1 & 4 & 4 & 8 & 8 & 9 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} x \leq \begin{bmatrix} 30 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8.0.2b}$$

$$x_i \in \{0, 1\} \text{ for } i \in \{1, 2, 3, 4, 5, 6\} \tag{8.0.2c}$$

9 | I

The code for solving the master problem and the sub problem can be found in the file "ExerciseGHI.jl" as solve_master and solve_sub respectively.
The iterations performed are seen in table 9.1. We use the simplification of representing the extreme point $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$ in the constraint $\lambda \leq |K|$, meaning we can start by solving the subproblem with duals π and κ equal to zero.
Every time the subproblem is solved, we get a new extreme point, and a maximum improvement from adding that extreme point, the reduced cost. If the reduced cost is zero or less, we are done, since this is a maximizing problem, and there is then no improvements to be had. The optimal solution is then the solution from the last solved master problem If the reduced cost is greater than zero, the extreme point is added to the matrix $\bar{X}_1$ for the next iteration of the master problem.
Every time the master problem is solved, we get new duals π and κ, which are then fed to the next iteration of the subproblem.
The first found extreme point, $\begin{bmatrix}0 & 0 & 1 & 1 & 0 & 1\end{bmatrix}$, can be interpreted as a knapsack with both items from class 2. The second extreme point, $\begin{bmatrix}1 & 1 & 0 & 0 & 1 & 0\end{bmatrix}$, can be interpreted as a knapsack with both items from class 1. The final solution has $\lambda = \begin{bmatrix}1 & 1\end{bmatrix}$, so both extreme points are used once. Then, it represents one knapsack with both items from class 1, and one knapsack with both items from class 2. The two possible, symmetric solutions with the original variable naming are then:

$$x_{111} = 1 \quad x_{121} = 1 \quad y_{11} = 1 \quad x_{212} = 1 \quad x_{222} = 1 \quad y_{22} = 1 \tag{9.0.1}$$

And

$$x_{211} = 1 \quad x_{221} = 1 \quad y_{21} = 1 \quad x_{112} = 1 \quad x_{122} = 1 \quad y_{12} = 1 \tag{9.0.2}$$

We get an integer solution to the Danzig-Wolfe relaxation, meaning that we have found the optimal solution to the problem.

Problem	Input	Output
Subproblem	$\pi = \begin{bmatrix}0 & 0\end{bmatrix}$ $\kappa = 0$	Reduced Cost = 2 Extreme point = $\begin{bmatrix}0 & 0 & 1 & 1 & 0 & 1\end{bmatrix}$
Master problem	$\bar{X}_1 = \begin{bmatrix}0 & 0 & 1 & 1 & 0 & 1\end{bmatrix}$	$\pi = \begin{bmatrix}0 & 2\end{bmatrix}$ $\kappa = 0$
Subproblem	$\pi = \begin{bmatrix}0 & 2\end{bmatrix}$ $\kappa = 0$	Reduced Cost = 1 Extreme point = $\begin{bmatrix}1 & 1 & 0 & 0 & 1 & 0\end{bmatrix}$
Master problem	$\bar{X}_1 = \begin{bmatrix}0 & 0 & 1 & 1 & 0 & 1\\ 1 & 1 & 0 & 0 & 1 & 0\end{bmatrix}$	$\pi = \begin{bmatrix}1 & 2\end{bmatrix}$ $\kappa = 0$
Subproblem	$\pi = \begin{bmatrix}1 & 2\end{bmatrix}$ $\kappa = 0$	Reduced Cost = 0 Extreme point = $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$
Results		$\lambda = \begin{bmatrix}1 & 1\end{bmatrix}$ $x_{111} = 1 \quad x_{121} = 1 \quad y_{11} = 1$ $x_{212} = 1 \quad x_{222} = 1 \quad y_{22} = 1$ Objective value = 3

Table 9.1 – Iterations when solving mini-instance2 using the identical subproblems simplification keeping constraint 4 in the master problem. Note that the final results would be equally valid if the items were assigned to the opposite knapsack, since the two knapsacks are identical.

10 | J

Our study numbers are 174356 and 184482 so

$$((174356 + 184482) \mod 5) + 1 = 4$$

Hence we have gotten the paper "Branch-and-price: Column Generation for Solving Huge Integer Programs". The paper is a broad review of the technique Branch-and-Price which applies column generation at each node and is also teached in this course. Often one also see that a Danzig-Wolfe reformulation is used in the scheme as we also do in the course.

In section 1 of the paper the Danzig-Wolfe reformulation is illustrated on two partitioning problems. The reason to use the Danzig-Wolfe reformulation is to obtain a tighter LP relaxation as mentioned on page 2 in the paper. We know from [1] that in general there are three things which makes a MILP hard to solve. The number of variables, the number of constraints and the integrality gap. The column generation provides us a way to handle a huge number of variables and the Danzig-Wolfe reformulation offers us a way to trade integrality gap to variables for some problems. This is also teached in lecture 1-3 in the course.

In the start of section 2 they walk through how the Danzig-Wolfe reformulation relies on the Minkowski-Weyl's Theorem and how to alter the procedure to handle identical subproblems. This is teached in lecture 1 and 4. In the last part of section 2 they discuss problem families and the application of Danzig-Wolfe and column generation to these families. They describe how Branch-and-Price is especially effective on partitioning and covering models because the reduction of the integrality gap is very effective here. Further they describe how symmetry in identical subproblems can be handled and if a problem can be formulated through both a partitioning and a covering then the covering is preferred.
If the model has a non-decomposable structure then the column generation can still offer a way to handle a huge number of variables but the Danzig-Wolfe reformulation will not be as effective as for the partitioning and covering models. A general discussion like this has not been presented in the lectures but the way to handle symmetry in partitioning and covering models has been presented in lecture 4.

Section 3 and 4 discuss Lagrangian duality and branching strategies. These we will not get further into here but proceed to section 5 in the interest of space. In section 5 computational issues are discussed which is at most new material compared to the course. The first two subsections discuss initial solutions and ways to trade of computation time and sub-optimal reduced costs in the column generation. In the next two subsections ways to mitigate the tailing-off effect that many column generation schemes exhibits are discussed. Especially section 5.4 is similarly to the dual stabilization which was discussed in lecture 5. In the paper they suggest to use an interior point algorithm instead of a simplex because the simplex always chooses an extreme point in one of the vertices in a potentially whole face of possible dual solutions. Oppositely the interior point algorithm will choose a point in the center of the face giving more stable dual values like the technique in lecture 5. Lastly different strategies for finding LP solution, some heuristics if one is interested in a good solution but not necessarily the optimal solution and the incorporation of row generation also known as the cutting plane method is discussed.

Bibliography

[1] L. N. Hoang. Column generation and dantzig-wolfe decomposition. [Online]. Available: http://www.science4all.org/article/column-generation/