

DTU42136, Large Scale Optimization using Decomposition

Assignment 1 Appendix, OptiGas: Optimize Gas Network using Benders Decomposition

Edward J. Xu (Jie Xu), s181238, DTU Management
March 20th, 2019

1 Appendix

Function file "BendersMilp_EDXU" to calculate mixed integer linear problem using Benders Decomposition:

```
1 # Benders Algorithm for MILP with Sub and Ray Problems
2 # Version: 4.0
3 # Author: Edward J. Xu, edxu96@outlook.com
4 # Date: March 20th, 2019
5 module BendersMilp_EDXU
6     export BendersMilp
7     using JuMP
8     using GLPKMathProgInterface
9     using PrettyTables
10    function BendersMilp(; n_x, n_y, vec_max_y, vec_c, vec_f, vec_b, mat_a, mat_b, epsilon, timesIterationMax)
11        println("-----\n",
12                "1/4. Begin Optimization -----",
13                "\n")
14        # Define Master problem
15        n_constraint = length(mat_a[:, 1])
16        model_mas = Model(solver = GLPKSolverMIP())
17        @variable(model_mas, q)
18        @variable(model_mas, vec_y[1: n_y] >= 0, Int)
19        @objective(model_mas, Min, (transpose(vec_f) * vec_y + q)[1])
20        @constraint(model_mas, vec_y[1: n_y] .<= vec_max_y)
21
22
23    function solve_master(vec_uBar, opt_cut::Bool)
24        if opt_cut
25            @constraint(model_mas, (transpose(vec_uBar) * (vec_b - mat_b * vec_y))[1] <= q)
26        else # Add feasible cut Constraints
27            @constraint(model_mas, (transpose(vec_uBar) * (vec_b - mat_b * vec_y))[1] <= 0)
28        end
29        @constraint(model_mas, (transpose(vec_uBar) * (vec_b - mat_b * vec_y))[1] <= q)
30        solve(model_mas)
31        vec_result_y = getvalue(vec_y)
32        # println("vec_y: $vec_result_y")
33        return getobjectivevalue(model_mas)
34    end
35
36
37    function solve_sub(vec_uBar, vec_yBar, n_constraint, vec_b, mat_b, mat_a, vec_c)
38        model_sub = Model(solver = GLPKSolverLP())
39        @variable(model_sub, vec_u[1: n_constraint] >= 0)
40        @objective(model_sub, Max, (transpose(vec_b - mat_b * vec_yBar) * vec_u)[1])
41        constraintsForDual = @constraint(model_sub, transpose(mat_a) * vec_u .<= vec_c)
42        solution_sub = solve(model_sub)
43        print("----- Sub Problem -----", "\n") # , model_sub)
44        vec_uBar = getvalue(vec_u)
45        if solution_sub == :Optimal
46            vec_result_x = zeros(length(vec_c))
47            vec_result_x = getdual(constraintsForDual)
48            return (true, getobjectivevalue(model_sub), vec_uBar, vec_result_x)
49        end
50        if solution_sub == :Unbounded
51            print("Not solved to optimality because feasible set is unbounded.\n")
52            return (false, getobjectivevalue(model_sub), vec_uBar, repeat([NaN], length(vec_c)))
53        end
54    end
55
56
57    function solve_ray(vec_uBar, vec_yBar, n_constraint, vec_b, mat_b, mat_a)
58        # model_ray = Model(solver = GurobiSolver())
59        model_ray = Model(solver = GLPKSolverLP())
60        @variable(model_ray, vec_u[1: n_constraint] >= 0)
61        @objective(model_ray, Max, 1)
62        @constraint(model_ray, (transpose(vec_b - mat_b * vec_yBar) * vec_u)[1] == 1)
63        @constraint(model_ray, transpose(mat_a) * vec_u .<= 0)
64        solve(model_ray)
65        print("----- Ray Problem -----", "\n") # , model_ray)
66        vec_uBar = getvalue(vec_u)
67        obj_ray = getobjectivevalue(model_ray)
68        return (obj_ray, vec_uBar)
69    end
70    end
```

```

71
72 # Begin Calculation
73 let
74     boundUp = Inf
75     boundLow = - Inf
76     epsilon = 0
77     # initial value of master variables
78     vec_uBar = zeros(n_constraint, 1)
79     vec_yBar = zeros(n_y, 1)
80     vec_result_x = length(n_x)
81     dict_obj_mas = Dict()
82     dict_q = Dict()
83     dict_obj_sub = Dict()
84     dict_obj_ray = Dict()
85     dict_boundUp = Dict()
86     dict_boundLow = Dict()
87     obj_sub = 0
88     timesIteration = 1
89     while ((boundUp - boundLow > epsilon) && (timesIteration <= timesIterationMax))
90     # while ((!(boundUp - boundLow < epsilon) && (result_q != obj_sub))) && (timesIteration <= timesIterationMax))
91         (bool_solutionSubModel, obj_sub, vec_uBar, vec_result_x) = solve_sub(vec_uBar, vec_yBar, n_constraint,
92                                     vec_b, mat_b, mat_a, vec_c)
93
94         if bool_solutionSubModel
95             boundUp = min(boundUp, obj_sub + (transpose(vec_f) * vec_yBar)[1])
96         else
97             (obj_ray, vec_uBar) = solve_ray(vec_uBar, vec_yBar, n_constraint, vec_b, mat_b, mat_a)
98         end
99         obj_mas = solve_master(vec_uBar, bool_solutionSubModel)
100         vec_yBar = getvalue(vec_y)
101         # println("vec_yBar: $vec_yBar")
102         boundLow = max(boundLow, obj_mas)
103         dict_boundUp[timesIteration] = boundUp
104         dict_boundLow[timesIteration] = boundLow
105         if bool_solutionSubModel
106             dict_obj_mas[timesIteration] = obj_mas
107             dict_obj_sub[timesIteration] = obj_sub
108             result_q = getvalue(q)
109             dict_q[timesIteration] = result_q
110             println("_____ Result in $(timesIteration)-th Iteration with Sub ",
111                     "_____~n", "boundUp: $(round(boundUp, digits = 5)), ",
112                     "boundLow: $(round(boundLow, digits = 5)), obj_mas: $(round(obj_mas, digits = 5)), ",
113                     "q: $result_q, obj_ray: $(round(obj_ray, digits = 5)).")
114             vec_result_y = getvalue(vec_y)
115             println("vec_y: $vec_result_y")
116         else
117             dict_obj_mas[timesIteration] = obj_mas
118             dict_obj_ray[timesIteration] = obj_ray
119             result_q = getvalue(q)
120             dict_q[timesIteration] = result_q
121             println("_____ Result in $(timesIteration)-th Iteration with Ray ",
122                     "_____~n", "boundUp: $(round(boundUp, digits = 5)), ",
123                     "boundLow: $(round(boundLow, digits = 5)), obj_mas: $(round(obj_mas, digits = 5)), ",
124                     "q: $result_q, obj_ray: $(round(obj_ray, digits = 5)).")
125             vec_result_y = getvalue(vec_y)
126             println("vec_y: $vec_result_y")
127         end
128         timesIteration += 1
129     end # _____
130     # @constraint(model_mas, q == obj_sub) # ???
131     # solve(model_mas) # ???
132     println("obj_mas: $(getobjectivevalue(model_mas))")
133     println("_____ Master Problem _____~n")
134     println(model_mas)
135     println("_____~n",
136             "_____ 2/4. Result _____~n",
137             "_____")
138     println("boundUp: $(round(boundUp, digits = 5)), boundLow: $(round(boundLow, digits = 5)), ",
139             "difference: $(round(boundUp - boundLow, digits = 5))")
140     println("vec_x: $vec_result_x")
141     vec_result_y = getvalue(vec_y)
142     result_q = getvalue(q)
143     println("vec_y: $vec_result_y")
144     println("result_q: $result_q")
145     println("_____~n",
146             "_____ 3/4. Iteration Result _____~n",
147             "_____")
148     # Initialize
149     seq_timesIteration = collect(1: (timesIteration - 1))
150     vec_boundUp = zeros(timesIteration - 1)
151     vec_boundLow = zeros(timesIteration - 1)
152     vec_obj_subRay = zeros(timesIteration - 1)
153     vec_obj_mas = zeros(timesIteration - 1)
154     vec_q = zeros(timesIteration - 1)

```

```

154     vec_type = repeat(["ray"], (timesIteration - 1))
155     #
156     for i = 1: (timesIteration - 1)
157         vec_obj_mas[i] = round(dict_obj_mas[i], digits = 5)
158         vec_boundUp[i] = round(dict_boundUp[i], digits = 5)
159         vec_boundLow[i] = round(dict_boundLow[i], digits = 5)
160         vec_q[i] = round(dict_q[i], digits = 5)
161         if haskey(dict_obj_sub, i)
162             vec_type[i] = "sub"
163             vec_obj_subRay[i] = round(dict_obj_sub[i], digits = 5)
164         else
165             vec_obj_subRay[i] = round(dict_obj_ray[i], digits = 5)
166         end
167     end
168     table_iterationResult = heatmap(seq_timesIteration, vec_boundUp, vec_boundLow,
169                                     vec_obj_mas, vec_q, vec_type, vec_obj_subRay)
170     pretty_table(table_iterationResult,
171                  ["Seq", "boundUp", "boundLow", "obj_mas", "q", "type_sub", "obj_sub/ray"],
172                  compact; alignment=:l)
173 end
174 # return (vec_result_x, vec_result_y, dict_obj_mas, dict_obj_sub, dict_obj_ray)
175 println("
176         _____ 4/4. Nominal Ending _____\n",
177         "\n")
178 end
179 end

```

File "OptiGas_DTU42136_EDXU" to calculate the assignment:

```

1 # OptiGas: Optimize Gas Network using Benders Algorithm
2 # Edward J. Xu, edxu96@outlook.com
3 # March 20th, 2019
4 push!(LOAD_PATH, "$(homedir())/Desktop/OptiGas, DTU42136")
5 cd("$(homedir())/Desktop/OptiGas, DTU42136")
6 using BendersMilp_EDXU
7 using LinearAlgebra
8 # -----
9 # 1. Parameters
10 vec_nameNodes = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
11 numNodes = length(vec_nameNodes)
12 vec_xNode = [1 3 8 2 4 5 6 7 9 9]
13 vec_yNode = [1 2 1 7 9 5 3 7 9 4]
14 mat_arcTwoNodes = [0 1 0 0 0 0 0 0 0 0;
15                    0 0 0 0 0 0 1 0 0 0;
16                    0 0 0 0 0 0 0 0 0 1;
17                    1 0 0 0 0 1 0 0 0 0;
18                    0 0 0 0 0 1 0 0 0 0;
19                    1 0 0 1 0 0 0 1 0 0;
20                    0 1 1 0 0 0 0 0 0 0;
21                    0 0 0 0 0 0 0 0 1 0;
22                    0 0 0 0 1 0 0 0 0 0;
23                    0 0 0 0 0 0 1 1 0 0]
24 vec_netEject = [0 0 0 0 0 90 0 0 0 80]
25 vec_netInject = [10 30 20 10 40 0 25 20 15 0]
26 mat_distance = zeros(Float64, numNodes, numNodes)
27 for m = 1: numNodes
28     for n = 1: numNodes
29         mat_distance[m, n] = floor(sqrt((vec_xNode[m] - vec_xNode[n]) * (vec_xNode[m] - vec_xNode[n]) +
30                                         (vec_yNode[m] - vec_yNode[n]) * (vec_yNode[m] - vec_yNode[n])))
31     end
32 end
33 mat_fixedCost = zeros(Float64, numNodes, numNodes)
34 for m = 1: numNodes
35     for n = 1: numNodes
36         mat_fixedCost[m, n] = 10 * mat_distance[m, n]
37     end
38 end
39 # -----
40 # Question 4
41 # Formulation of Matrix A
42 mat_a1_1 = zeros(numNodes^2, numNodes^2)
43 for mn = 1: numNodes^2
44     mat_a1_1[mn, mn] = -1
45 end
46 mat_a2_1 = zeros(numNodes^2, numNodes^2)
47 mat_a3_1 = zeros(numNodes, numNodes^2)
48 seq_zeroTenNinety = collect(0:10:90)
49 for m = 1: numNodes
50     mat_a3_1[m, (10 * m - 9): (10 * m)] = repeat([-1], 10) # sent out
51     mat_a3_1[m, (seq_zeroTenNinety + repeat([m], 10))] = repeat([1], 10) # sent in
52     mat_a3_1[:, (11 * m - 10)] = repeat([0], 10) # self-sending doesn't count
53 end
54 mat_a_1 = vcat(mat_a1_1, mat_a2_1, mat_a3_1)

```

```

55 # Formulation of Matrix B
56 mat_b1_1 = zeros(numNodes^2, numNodes^2)
57 for mn = 1: numNodes^2
58     mat_b1_1[mn, mn] = 170
59 end
60 mat_b2_1 = zeros(numNodes^2, numNodes^2)
61 for mn = 1: numNodes^2
62     mat_b2_1[mn, mn] = 1
63 end
64 # mat_b2_1 = Diagonal(repeat([1], numNodes^2))
65 mat_b3_1 = zeros(numNodes, numNodes^2)
66 mat_b_1 = vcat(mat_b1_1, mat_b2_1, mat_b3_1)
67 # Formulation of Vector b
68 vec_b1_1 = zeros(numNodes^2)
69 vec_b2_1 = zeros(numNodes^2)
70 for m = 1: numNodes
71     for n = 1: numNodes
72         vec_b2_1[10 * (m - 1) + n] = mat_arcTwoNodes[m, n]
73     end
74 end
75 vec_b3_1 = zeros(numNodes)
76 for n = 1: numNodes
77     vec_b3_1[n] = vec_netInject[n] - vec_netEject[n]
78 end
79 vec_b_1 = vcat(vec_b1_1, vec_b2_1, vec_b3_1)
80 vec_b_1 = hcat(vec_b_1)
81 #
82 vec_max_y_1 = repeat([1], numNodes^2)
83 vec_max_y_1 = hcat(vec_max_y_1)
84 #
85 vec_c_1 = zeros(numNodes^2)
86 for m = 1: numNodes
87     for n = 1: numNodes
88         vec_c_1[10 * (m - 1) + n] = mat_distance[m, n]
89     end
90 end
91 vec_c_1 = hcat(vec_c_1)
92 #
93 vec_f_1 = zeros(numNodes^2)
94 for m = 1: numNodes
95     for n = 1: numNodes
96         vec_f_1[10 * (m - 1) + n] = mat_fixedCost[m, n]
97     end
98 end
99 vec_f_1 = hcat(vec_f_1)
100 #
101 BendersMilp(n_x = numNodes^2,
102             n_y = numNodes^2,
103             vec_max_y = vec_max_y_1,
104             vec_c = vec_c_1,
105             vec_f = vec_f_1,
106             vec_b = vec_b_1,
107             mat_a = mat_a_1,
108             mat_b = mat_b_1,
109             epsilon = 0.0001,
110             timesIterationMax = 1000)
111 # 1280 - sum(mat_arcTwoNodes .* mat_fixedCost)
112 #
113 # Question 5
114 # Formulation of Matrix A
115 mat_a1_2 = zeros(numNodes^2, numNodes^2)
116 for mn = 1: numNodes^2
117     mat_a1_2[mn, mn] = -1
118 end
119 mat_a3_2 = zeros(numNodes, numNodes^2)
120 seq_zeroTenNinety = collect(0:10:90)
121 for m = 1: numNodes
122     mat_a3_2[m, (10 * m - 9):(10 * m)] = repeat([-1], 10) # sent out
123     mat_a3_2[m, (seq_zeroTenNinety + repeat([m], 10))] = repeat([1], 10) # sent in
124     mat_a3_2[:, (11 * m - 10)] = repeat([0], 10) # self-sending doesn't count
125 end
126 mat_a_2 = vcat(mat_a1_2, mat_a3_2)
127 # Formulation of Matrix B
128 mat_b1_2 = zeros(numNodes^2, numNodes^2)
129 for mn = 1: numNodes^2
130     mat_b1_2[mn, mn] = 170
131 end
132 mat_b3_2 = zeros(numNodes, numNodes^2)
133 mat_b_2 = vcat(mat_b1_2, mat_b3_2)
134 # Formulation of Vector b
135 vec_b1_2 = zeros(numNodes^2)
136 vec_b3_2 = zeros(numNodes)
137 for n = 1: numNodes

```

```

138     vec_b3_2[n] = vec_netInject[n] - vec_netEject[n]
139 end
140 vec_b_2 = vcat(vec_b1_2, vec_b3_2)
141 vec_b_2 = hcat(vec_b_2)
142 #
143 vec_max_y_2 = repeat([1], numNodes^2)
144 vec_max_y_2 = hcat(vec_max_y_2)
145 #
146 vec_c_2 = zeros(numNodes^2)
147 for m = 1: numNodes
148     for n = 1: numNodes
149         vec_c_2[10 * (m - 1) + n] = mat_distance[m, n]
150     end
151 end
152 vec_c_2 = hcat(vec_c_2)
153 #
154 vec_f_2 = zeros(numNodes^2)
155 for m = 1: numNodes
156     for n = 1: numNodes
157         vec_f_2[10 * (m - 1) + n] = mat_fixedCost[m, n]
158     end
159 end
160 vec_f_2 = hcat(vec_f_2)
161 #
162 BendersMilp(n_x = numNodes^2,
163             n_y = numNodes^2,
164             vec_max_y = vec_max_y_2,
165             vec_c = vec_c_2,
166             vec_f = vec_f_2,
167             vec_b = vec_b_2,
168             mat_a = mat_a_2,
169             mat_b = mat_b_2,
170             epsilon = 0.0001,
171             timesIterationMax = 1000)
172 # obj = sum(mat_arcTwoNodes .* mat_fixedCost)

```