National Technical University of Athens
School of Electrical & Computer Engineering

Semester Assignment Report

Course: Mathematical models programming (9^{th} Semester)

Professor: A. Papavasileiou

Author

Alexandros Spandonidis 03121159

January 8, 2026

Abstract

Η παρούσα εργασία αφορά την υλοποίηση γνωστών αλγορίθμων αποσύνθεσης και την εφαρμογή τους στην επίλυση του προβλήματος επέκτασης συστήματος. Πιο συγκεκριμένα, υλοποιούνται: (α) ο αλγόριθμος αποσύνθεσης Benders στο πρόβλημα επέκτασης συστήματος, (β) η μέθοδος σχήματος L στο στοχαστικό πρόβλημα επέκτασης συστήματος και (γ) η μέθοδος σχήματος L πολλαπλών τομών στο ίδιο στοχαστικό πρόβλημα. Παρουσιάζονται τα δεδομένα, οι μαθηματικές διατυπώσεις, καθώς και τα αποτελέσματα σύγκλισης μέσω της εξέλιξης των αποφάσεων επένδυσης ανά επανάληψη. Όλοι οι κώδικες, πίνακες και δεδομένα είναι διαθέσιμα στο GitHub.

`https://github.com/alexspando/montela-mathimatikou-programmatismou`

Αλγόριθμος αποσύνθεσης Benders

Δεδομένα προβλήματος και καμπύλη διάρκειας φορτίου

Η καμπύλη διάρκειας φορτίου αναπαρίσταται μέσω οριζόντιας διαστρωμάτωσης (slicing) και παρουσιάζεται στον Πίνακα 1.

Τα κόστη τεχνολογιών (λειτουργικό και αρχικής επένδυσης) παρατίθενται στον Πίνακα 2. Για τη μη εξυπηρετούμενη ζήτηση (load not served) χρησιμοποιείται τιμή $V = 1000$ €/MWh.

Μαθηματική διατύπωση (ντετερμινιστική περίπτωση)

Θεωρούμε n τεχνολογίες και m οριζόντιες φέτες ζήτησης. Οι μεταβλητές είναι:

- $x_i \geq 0$: επένδυση (MW) στην τεχνολογία i,
- $p_{ij} \geq 0$: παραγωγή (MW) τεχνολογίας i στη φέτα j,
- $lol_j \geq 0$: μη εξυπηρετούμενο φορτίο (MW) στη φέτα j.

Το πρόβλημα γράφεται ως:

$$\min_{p,lol,x} \quad \sum_{i=1}^{n} I_i x_i + \sum_{j=1}^{m}\sum_{i=1}^{n} MC_i\, T_j\, p_{ij} + \sum_{j=1}^{m} V\, T_j\, lol_j \tag{1}$$

$$\text{υπό τους περιορισμούς} \quad \sum_{i=1}^{n} p_{ij} + lol_j = \Delta D_j, \quad j = 1, \ldots, m, \tag{2}$$

$$\sum_{j=1}^{m} p_{ij} \leq x_i, \quad i = 1, \ldots, n, \tag{3}$$

$$p_{ij} \geq 0,\ lol_j \geq 0,\ x_i \geq 0. \tag{4}$$

Δομή αποσύνθεσης Benders

Το πρόβλημα αφέντη (master) είναι:

$$\min_{x,\theta} \quad \sum_{i=1}^{n} I_i x_i + \theta \tag{5}$$

$$\text{υπό} \quad \theta \geq \sum_{j=1}^{m} \lambda_j^{(v)} \, \Delta D_j + \sum_{i=1}^{n} \rho_i^{(v)} \, x_i, \quad v = 1, \ldots, k, \tag{6}$$

$$x \geq 0, \; \theta \geq 0, \tag{7}$$

όπου $\lambda_j^{(v)}, \rho_i^{(v)}$ είναι οι (βέλτιστοι) δυϊκοί πολλαπλασιαστές του προβλήματος σκλάβου (slave) που αντιστοιχεί σε δεδομένη λύση $x^{(k)}$.

Το πρόβλημα σκλάβο (S) για δοσμένο $x^{(k)}$:

$$\min_{p,lol} \quad \sum_{j=1}^{m} \sum_{i=1}^{n} MC_i \, T_j \, p_{ij} + \sum_{j=1}^{m} V \, T_j \, lol_j \tag{8}$$

$$\text{υπό} \quad \sum_{i=1}^{n} p_{ij} + lol_j = \Delta D_j, \quad j = 1, \ldots, m, \tag{9}$$

$$\sum_{j=1}^{m} p_{ij} \leq x_i^{(k)}, \quad i = 1, \ldots, n, \tag{10}$$

$$p \geq 0, \; lol \geq 0. \tag{11}$$

Υλοποίηση σε Julia (Benders decomposition)

Τα αγγλικά τμήματα/σχόλια/κεφαλίδες παραμένουν ως έχουν (English). Ο κώδικας που ακολουθεί είναι αυτός που δόθηκε στην αναφορά.

```
################################################################
# BENDERS DECOMPOSITION FOR POWER SYSTEM EXPANSION PLANNING
#
# - Slave      optimality cuts
# -   iteration: x, Q(x), LB, UB, gap, duals
# -  results  benders_results.csv
################################################################
using JuMP
using GLPK
using CSV
using DataFrames
using Printf
import MathOptInterface as MOI

function run_benders()
    println(=== LOADING INPUT DATA ===)
    ##############################
    # Load Needs Data (Slices) #
    ##############################
    needs = CSV.read(needs.csv, DataFrame)
    #    (..  duration -> duration)
```

```
rename!(needs, Symbol.(strip.(String.(names(needs)))))
println(\nNeeds:)
println(needs)

categories = needs[!, 1]
T = Float64.(needs[!, 2])
minL = Float64.(needs[!, 3])
maxL = Float64.(needs[!, 4])
m = length(categories)

# D_j = max_level - min_level
D = [maxL[j] - minL[j] for j in 1:m]
println(\nT (hours per slice) = , T)
println(D per slice = , D)

#######################################
# Load Technologies (MC and Inv cost) #
#######################################
tech = CSV.read(technology.csv, DataFrame)
rename!(tech, Symbol.(strip.(String.(names(tech)))))
println(\nTechnologies:)
println(tech)

names_tech = String.(tech[!, 1]) # technology
MC = Float64.(tech[!, 2])        # cost
I  = Float64.(tech[!, 3])        # initial_investment
n = length(names_tech)

println(\nMC = , MC)
println(I = , I)
println(Technologies = , names_tech)

V = 1000.0 # /MWh

#############################################################
# MASTER PROBLEM (Investment problem with  & Benders cuts)
#############################################################
master = Model(GLPK.Optimizer)
@variable(master, x[1:n] >= 0) # investments (MW)
@variable(master,  >= 0)
@objective(master, Min, sum(I[i] * x[i] for i in 1:n) + )

println(\n=== STARTING BENDERS ITERATIONS ===)
max_iters = 50
tol = 1e-3
LB = -Inf
UB = +Inf

#   table:
res = DataFrame()
res.iter = Int[]
```

```
for nm in names_tech
    res[!, Symbol(nm)] = Float64[]
end
res.Q = Float64[]
res.LB = Float64[]
res.UB = Float64[]
res.gap = Float64[]
res.cut_type = String[]

##############################################################
# MAIN LOOP
##############################################################
for k in 1:max_iters
    println(\n----------------------------)
    println(Iteration $k)
    println(----------------------------)

    ##########################
    # 1) Solve MASTER
    ##########################
    optimize!(master)
    mst_status = termination_status(master)
    if mst_status != MOI.OPTIMAL
        println(Master not optimal. status = $mst_status. Stopping.)
        break
    end

    xk = value.(x)
    k = value()
    invest_cost = sum(I[i] * xk[i] for i in 1:n)

    @printf([Master] obj = %.6e\n, objective_value(master))
    @printf([Master]  = %.6e\n, k)
    @printf([Master] Inv = %.6e\n, invest_cost)
    println([Master] x (MW):)
    for i in 1:n
        @printf( %-10s : %.4f\n, names_tech[i], xk[i])
    end

    LB = objective_value(master)

    ##########################
    # 2) Solve SLAVE
    ##########################
    sub = Model(GLPK.Optimizer)
    @variable(sub, p[1:n, 1:m] >= 0)   # dispatch (MW)
    @variable(sub, lol[1:m] >= 0)      # unserved (MW)

    @objective(sub, Min,
        sum(MC[i] * T[j] * p[i,j] for i in 1:n, j in 1:m) +
        sum(V * T[j] * lol[j] for j in 1:m)
```

```
    )

    # Demand with LOL (always feasible)
    @constraint(sub, demand[j=1:m],
        sum(p[i,j] for i in 1:n) + lol[j] == D[j]
    )

    # Capacity limits coupled with xk
    @constraint(sub, cap[i=1:n],
        sum(p[i,j] for j in 1:m) <= xk[i]
    )

    optimize!(sub)
    sub_status = termination_status(sub)

    if sub_status == MOI.INFEASIBLE
        #  lol   ,      .
        println([Slave] INFEASIBLE (unexpected with LOL). Adding a simple
feasibility cut.)
        #  feasibility cut (    lol)
        @constraint(master, sum(x[i] for i in 1:n) >= maximum(D))

        #    results
        push!(res.iter, k)
        for i in 1:n
            push!(res[!, Symbol(names_tech[i])], xk[i])
        end
        push!(res.Q, NaN)
        push!(res.LB, LB)
        push!(res.UB, UB)
        push!(res.gap, UB - LB)
        push!(res.cut_type, feasibility)
        continue

    elseif sub_status != MOI.OPTIMAL
        println([Slave] status = $sub_status. Stopping.)
        break
    end

    Qx = objective_value(sub)
    @printf([Slave] Q(x) = %.6e\n, Qx)

    UB = min(UB, invest_cost + Qx)
    gap = UB - LB
    @printf([Bounds] LB = %.6e | UB = %.6e | gap = %.6e\n, LB, UB, gap)

    # Dual multipliers
     = dual.(demand) # one per slice
     = dual.(cap)    # one per technology
    println(Dual  (per slice) = , )
    println(Dual  (per tech) = , )
```

```
        #  iteration
        push!(res.iter, k)
        for i in 1:n
            push!(res[!, Symbol(names_tech[i])], xk[i])
        end
        push!(res.Q, Qx)
        push!(res.LB, LB)
        push!(res.UB, UB)
        push!(res.gap, gap)
        push!(res.cut_type, optimality)

        ##########################
        # Convergence check
        ##########################
        if abs(gap) <= tol
            println(\n>>> CONVERGED (gap  $(tol)) <<<)
            break
        end

        ##########################
        # 3) Add Benders optimality cut
        #
        #   _j _j D_j + _i _i x_i
        #
        # ( x_i     master,   xk.)
        ##########################
        @constraint(master,
             >= sum([j] * D[j] for j in 1:m) + sum([i] * x[i] for i in 1:n)
        )
        println(Added optimality cut.)
    end

    println(\n===== FINAL SOLUTION =====)
    xstar = value.(x)
    println(x* (MW):)
    for i in 1:n
        @printf( %-10s : %.4f\n, names_tech[i], xstar[i])
    end
    @printf(* = %.6e\n, value())
    @printf(Final LB = %.6e | Final UB = %.6e | gap = %.6e\n, LB, UB, UB - LB)

    println(\nResults table:)
    println(res)
    CSV.write(benders_results.csv, res)
    println(\nSaved: benders_results.csv)
end

# Run when you include the file
run_benders()
```

Τα δεδομένα του προβλήματος αποθηκεύονται στα αρχεία `technology.csv` και `needs.csv`,

τα οποία περιέχουν αντίστοιχα τα κόστη τεχνολογιών και τις ανάγκες/φέτες του συστήματος. Τα αποτελέσματα της αποσύνθεσης αποθηκεύονται στο `benders_results.csv`. Για λόγους συνοπτικής παρουσίασης, εδώ παρατίθεται η εξέλιξη των αποφάσεων πρώτου σταδίου (επενδύσεις σε MW) μέχρι τη σύγκλιση. Στη δομή του προβλήματος, το υποπρόβλημα είναι πάντα εφικτό λόγω της μεταβλητής *lol*, συνεπώς όλες οι τομές είναι τομές βελτιστότητας.

Εξέλιξη επενδύσεων (Benders)

Table 3: Εξέλιξη επενδύσεων ανά επανάληψη για Benders.

iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	10701.6
2	0.0	0.0	0.0	21168.6
3	0.0	17978.7	0.0	0.0
4	13075.4	0.0	0.0	0.0
5	0.0	0.0	8989.4	2179.6
6	0.0	918.2	8989.4	750.7
7	0.0	918.2	8989.4	1576.9
8	0.0	0.0	10064.2	1104.8
9	1987.8	1664.7	7516.5	0.0
10	6460.5	0.0	3158.4	1550.1
11	4111.3	3096.4	4628.2	0.0
12	1918.0	2165.0	7068.0	0.0

Όπως αναμένεται, στα πρώιμα στάδια ο αλγόριθμος τείνει να προτείνει επενδύσεις σε τεχνολογίες με χαμηλό αρχικό κόστος ανά MW (εν προκειμένω το πετρέλαιο). Καθώς ενσωματώνεται προοδευτικά η πληροφορία του λειτουργικού κόστους μέσω των τομών βελτιστότητας, η λύση μεταβάλλεται μέχρι να προκύψει μίγμα τεχνολογιών που ελαχιστοποιεί το συνολικό κόστος (επένδυσης και λειτουργίας).

Η μέθοδος σχήματος L

Σενάρια ζήτησης και πιθανότητες

Εξετάζεται στοχαστική εκδοχή του προβλήματος με δύο σενάρια ζήτησης: (i) σενάριο αναφοράς και (ii) σενάριο *10x Αιολικά*. Η περιγραφή των φετών φορτίου ανά σενάριο παρουσιάζεται στον Πίνακα 4. Το σενάριο αναφοράς έχει πιθανότητα 10%, ενώ το σενάριο *10x Αιολικά* πιθανότητα 90%.

Στοχαστική διατύπωση

Το στοχαστικό πρόβλημα δύο σταδίων γράφεται:

$$\min_{x,p,lol} \quad \sum_{i=1}^{n} I_i x_i + \sum_{\omega\in\Omega} P_\omega \left(\sum_{j=1}^{m}\sum_{i=1}^{n} MC_i\, T_j\, p_{ij\omega} + \sum_{j=1}^{m} V\, T_j\, lol_{j\omega} \right) \tag{12}$$

$$\text{υπό} \quad \sum_{i=1}^{n} p_{ij\omega} + lol_{j\omega} = \Delta D_{j\omega}, \quad j = 1,\dots,m,\ \omega\in\Omega, \tag{13}$$

$$\sum_{j=1}^{m} p_{ij\omega} \le x_i, \quad i = 1,\dots,n,\ \omega\in\Omega, \tag{14}$$

$$x \ge 0,\ p \ge 0,\ lol \ge 0. \tag{15}$$

Το πρόβλημα αφέντη είναι πανομοιότυπο με εκείνο της αποσύνθεσης Benders, ενώ το πρόβλημα σκλάβο για κάθε σενάριο ω δίνεται από:

$$(S_\omega): \quad \min_{p,lol} \quad \sum_{j=1}^{m}\sum_{i=1}^{n} MC_i\, T_j\, p_{ij} + \sum_{j=1}^{m} V\, T_j\, lol_j \tag{16}$$

$$\text{υπό} \quad \sum_{i=1}^{n} p_{ij} + lol_j = \Delta D_{j\omega}, \quad j = 1,\dots,m, \tag{17}$$

$$\sum_{j=1}^{m} p_{ij} \le x_i^{(k)}, \quad i = 1,\dots,n, \tag{18}$$

$$p \ge 0,\ lol \ge 0, \tag{19}$$

όπου $x^{(k)}$ είναι η τρέχουσα λύση επένδυσης από το master.

Εξέλιξη επενδύσεων (L-shaped)

Παρατηρείται ότι οι υποψήφιες επενδύσεις κάθε επανάληψης είναι διαφορετικές από τις προηγούμενες, ενώ δεν είναι απαραίτητα μονοτονικές. Ενδεικτικά, σε συγκεκριμένη επανάληψη ο αλγόριθμος εξετάζει λύση χωρίς πυρηνικά, γεγονός που οδηγεί σε πολύ υψηλό κόστος δεύτερου σταδίου.

Table 5: Εξέλιξη επενδύσεων ανά επανάληψη για τη μέθοδο σχήματος L.

iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	8736.43
2	0.0	0.0	0.0	15998.9
3	0.0	14677.2	0.0	0.0
4	10674.3	0.0	0.0	0.0
5	10674.3	0.0	0.0	13329.4
6	0.0	163.796	7174.81	3830.39
7	0.0	3300.75	7868.25	0.0
8	3045.3	3588.83	4534.87	0.0

iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
9	8289.91	0.0	0.0	2699.94
10	5511.38	0.0	4424.12	1233.5
11	7632.01	1256.7	1558.78	721.513
12	4783.51	1256.7	4407.28	59.033
13	4711.09	2020.14	3458.93	871.833
14	4754.81	1559.29	4031.41	849.643
15	5042.68	367.201	5272.32	801.543
16	5054.93	640.406	4880.57	816.728
17	5085.0	1311.0	3919.0	854.0

Υλοποίηση σε Julia (L-shaped method)

Τα αγγλικά τμήματα/σχόλια/κεφαλίδες παραμένουν ως έχουν (English). Ο κώδικας που ακολουθεί είναι αυτός που δόθηκε στην αναφορά.

```
#################################################################
# L-SHAPED METHOD FOR STOCHASTIC SYSTEM EXPANSION
#################################################################
using JuMP
using GLPK
using CSV
using DataFrames
using Printf
println(=== LOADING INPUT DATA ===)
#######################################
# Load Technologies (MC and Inv cost) #
#######################################
tech = CSV.read(technology.csv, DataFrame)
println(\nTechnologies:)
println(tech)
names = String.(tech.technology)
MC = Float64.(tech.cost)
I = Float64.(tech.initial_investment)
n = length(names)
V = 1000.0
#######################################
# Scenarios (2 scenarios)
#######################################
T = [1.0, 0.79908675799, 0.17123287671] # durations (not stochastic)
m = length(T)
P = [0.10, 0.90] # probabilities
 = length(P)
# D[j, ] = max-min (width of slice j in scenario )
D = [
 7086.0 3919.0; # base
 1918.0 3410.0; # medium
 2165.0 2986.0 # peak
] # size m x
println(\nScenario probabilities P = , P)
```

```
println(D per (slice j, scenario ):)
for  in 1:
 println( =$ D = , D[:, ])
end
################################################################
# MASTER PROBLEM
################################################################
master = Model(GLPK.Optimizer)
@variable(master, x[1:n] >= 0) # investment in MW
@variable(master,  >= 0)
@objective(master, Min, sum(I[i] * x[i] for i in 1:n) + )
cuts = Vector{ConstraintRef}()
################################################################
# SLAVE SOLVER FOR ONE SCENARIO
################################################################
function solve_slave_one_scenario(xk::Vector{Float64}, ::Int,
 MC::Vector{Float64}, T::Vector{Float64},
 V::Float64, D::Matrix{Float64})
 n = length(MC)
 m = length(T)
 sub = Model(GLPK.Optimizer)
 set_silent(sub)
 @variable(sub, p[1:n, 1:m] >= 0)
 @variable(sub, lol[1:m] >= 0)
 @objective(sub, Min,
 sum(MC[i] * T[j] * p[i, j] for i in 1:n, j in 1:m) +
 sum(V * T[j] * lol[j] for j in 1:m)
 )
 @constraint(sub, demand[j=1:m],
 sum(p[i, j] for i in 1:n) + lol[j] == D[j, ]
 )
 @constraint(sub, cap[i=1:n],
 sum(p[i, j] for j in 1:m) <= xk[i]
 )
 optimize!(sub)
 if termination_status(sub) != MOI.OPTIMAL
 error(Slave not optimal for scenario =$ (should not happen with LOL).)
 end
 Q = objective_value(sub)
  = dual.(demand)
  = dual.(cap)
 return Q, ,
end
################################################################
# RESULTS TABLE (decision evolution)
################################################################
# We build it dynamically so the CSV has real tech names.
results = DataFrame()
results.iter = Int[]
for nm in names
 results[!, Symbol(nm)] = Float64[] # x per tech
```

```
end
results. = Float64[]
results.investment_cost = Float64[]
results.EQ = Float64[] # expected recourse
results.gap = Float64[]
results.cut_type = String[]
############################################################
# MAIN L-SHAPED LOOP
############################################################
println(\n=== STARTING L-SHAPED ITERATIONS ===)
max_iters = 50
UB = +Inf
LB = -Inf
for k in 1:max_iters
 println(\n--------------------------)
 println(Iteration $k)
 println(--------------------------)
 ########################
 # 1. Solve MASTER
 ########################
 optimize!(master)
 if termination_status(master) != MOI.OPTIMAL
 println(Master not optimal. Stopping.)
 break
 end
 xk = value.(x)
 k = value()
 investment_cost = sum(I[i] * xk[i] for i in 1:n)
 LB = objective_value(master)
 println([Master] x = , xk)
 println([Master] Investment cost = , investment_cost)
 println([Master]  = , k)
 println([Master] LB = , LB)
 ########################
 # 2. Solve ALL SLAVES => Expected recourse + aggregated cut
 ########################
 Q_vals = zeros()
 Qbar = 0.0
  = 0.0
  = zeros(n)
 for  in 1:
 Q, ,  = solve_slave_one_scenario(xk, , MC, T, V, D)
 Q_vals[] = Q
 Qbar += P[] * Q
  += P[] * sum([j] * D[j, ] for j in 1:m)
 for i in 1:n
 [i] += P[] * [i]
 end
 end
 println([Slaves] Q = , Q_vals,  E[Q(x)] = , Qbar)
 ########################
```

```
 # 3. Update UB and gap
 ##########################
 global UB
 UB = min(UB, investment_cost + Qbar)
 gap = UB - LB
 println([Bounds] UB = , UB)
 println([Bounds] Gap = , gap)
 ##########################
 # 4. Store results row
 ##########################
 row = Dict{Symbol, Any}()
 row[:iter] = k
 for i in 1:n
 row[Symbol(names[i])] = xk[i]
 end
 row[:] = k
 row[:investment_cost] = investment_cost
 row[:EQ] = Qbar
 row[:gap] = gap
 row[:cut_type] = optimality
 push!(results, row)
 # Optional: write CSV each iteration (so you can watch it live)
 CSV.write(lshaped_results.csv, results)
 ##########################
 # 5. Convergence check
 ##########################
 if abs(gap) < 1e-3
 println(\n>>> CONVERGED <<<)
 break
 end
 ##########################
 # 6. Add NEW L-SHAPED CUT
 ##########################
 cut = @constraint(master,
  >=  + sum([i] * x[i] for i in 1:n)
 )
 push!(cuts, cut)
 println(Added aggregated optimality cut.)
end
println(\n===== FINAL SOLUTION =====)
println(x* = , value.(x))
println(* = , value())
println(\nResults saved to lshaped_results.csv.)
#################################################################
# Display from CSV
#################################################################
println(\n=== DISPLAYING OUTPUT FROM CSV ===)
df = CSV.read(lshaped_results.csv, DataFrame)
println(df)
```

Η μέθοδος σχήματος L πολλαπλών τομών

Διατύπωση προβλήματος αφέντη (multi-cut)

Στη μέθοδο πολλαπλών τομών χρησιμοποιείται ξεχωριστή μεταβλητή θ_ω ανά σενάριο, και ο στόχος είναι:

$$\min_{x,\theta} \quad \sum_{i=1}^{n} I_i x_i + \sum_{\omega \in \Omega} P_\omega \theta_\omega \tag{20}$$

$$\text{υπό} \quad \theta_\omega \geq \sum_{j=1}^{m} \lambda_{j\omega}^{(v)} \Delta D_{j\omega} + \sum_{i=1}^{n} \rho_{i\omega}^{(v)} x_i, \quad \forall \omega \in \Omega,\ v = 1, \ldots, k, \tag{21}$$

$$x \geq 0,\ \theta_\omega \geq 0. \tag{22}$$

Η υλοποίηση παράγει το `lshaped_multicut_results.csv` με αντίστοιχη πληροφορία (επενδύσεις, φράγματα, χάσμα, δυϊκές μεταβλητές και τύπο τομών). Συγκριτικά, απαιτούνται 11 επαναλήψεις για σύγκλιση (χωρίς να μετράται η επανάληψη 0), έναντι 15 επαναλήψεων για τη μέθοδο σχήματος L. Επιπλέον, παρατηρείται ότι μη μηδενικό κόστος δεύτερου σταδίου εμφανίζεται ήδη από την επανάληψη 3.

Εξέλιξη επενδύσεων (multi-cut L-shaped)

Table 6: Εξέλιξη επενδύσεων ανά επανάληψη για τη μέθοδο σχήματος L πολλαπλών τομών.

iteration	Coal (MW)	Gas (MW)	Nuclear (MW)	Oil (MW)
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	10701.6
2	0.0	14310.4	0.0	0.0
3	10407.5	0.0	0.0	0.0
4	7155.19	5033.97	0.0	0.0
5	2328.72	7155.19	1410.03	0.0
6	1280.16	8518.73	1647.69	0.0
7	2101.89	3310.95	5756.16	0.0
8	8768.3	235.699	2291.52	0.0
9	6396.0	3919.0	1168.74	0.0
10	8232.42	2165.0	771.58	0.0
11	5085.0	1311.0	3919.0	854.0

Υλοποίηση σε Julia (Multi-cut L-shaped)

Τα αγγλικά τμήματα/σχόλια/κεφαλίδες παραμένουν ως έχουν (English). Ο κώδικας που ακολουθεί είναι αυτός που δόθηκε στην αναφορά.

```
############################################################
# MULTI-CUT L-SHAPED METHOD FOR STOCHASTIC SYSTEM EXPANSION (LP)
############################################################
using JuMP, GLPK, CSV, DataFrames
```

```
using Printf
println(=== LOADING INPUT DATA ===)
#######################################
# Load Technologies (MC and Inv cost) #
#######################################
tech = CSV.read(technology.csv, DataFrame)
println(\nTechnologies:)
println(tech)
names = String.(tech.technology)
MC = Float64.(tech.cost)
I = Float64.(tech.initial_investment)
n = length(names)
V = 1000.0
T = [1.0, 0.79908675799, 0.17123287671] # durations (not stochastic)
m = length(T)
P = [0.10, 0.90] # probabilities
 = length(P)
# D[j, ] = max-min (width of slice j in scenario )
D = [
 7086.0 3919.0; # base
 1918.0 3410.0; # medium
 2165.0 2986.0 # peak
] # size m x
println(\nScenario probabilities P = , P)
println(D per (slice j, scenario ):)
for  in 1:
 println( =$ D = , D[:, ])
end
#################################################################
# MASTER PROBLEM (MULTI-CUT)
#################################################################
master = Model(GLPK.Optimizer)
@variable(master, x[1:n] >= 0) # investment in MW
@variable(master, [1:] >= 0) # one  per scenario (multi-cut)
@objective(master, Min,
 sum(I[i] * x[i] for i in 1:n) + sum(P[] * [] for  in 1:)
)
# Store cuts per scenario if you want (optional)
cuts_by_scenario = [ConstraintRef[] for _ in 1:]
#################################################################
# SLAVE SOLVER FOR ONE SCENARIO
#################################################################
function solve_slave_one_scenario(xk::Vector{Float64}, ::Int,
 MC::Vector{Float64}, T::Vector{Float64},
 V::Float64, D::Matrix{Float64})
 n = length(MC)
 m = length(T)
 sub = Model(GLPK.Optimizer)
 set_silent(sub)
 @variable(sub, p[1:n, 1:m] >= 0)
 @variable(sub, lol[1:m] >= 0)
```

```
 @objective(sub, Min,
 sum(MC[i] * T[j] * p[i, j] for i in 1:n, j in 1:m) +
 sum(V * T[j] * lol[j] for j in 1:m)
 )
 @constraint(sub, demand[j=1:m],
 sum(p[i, j] for i in 1:n) + lol[j] == D[j, ]
 )
 @constraint(sub, cap[i=1:n],
 sum(p[i, j] for j in 1:m) <= xk[i]
 )
 optimize!(sub)
 if termination_status(sub) != MOI.OPTIMAL
 error(Slave not optimal for scenario =$ (should not happen with LOL).)
 end
 Q = objective_value(sub)
  = dual.(demand) # length m
  = dual.(cap)    # length n
 return Q, ,
end
################################################################
# RESULTS TABLE (decision evolution)
################################################################
results = DataFrame()
results.iter = Int[]
# x per tech
for nm in names
 results[!, Symbol(nm)] = Float64[]
end
#  per scenario
for  in 1:
 results[!, Symbol(_$())] = Float64[]
end
results.investment_cost = Float64[]
results.EQ = Float64[] # expected recourse (sum P*Q)
results.LB = Float64[]
results.UB = Float64[]
results.gap = Float64[]
results.cut_type = String[]
################################################################
# MAIN MULTI-CUT L-SHAPED LOOP
################################################################
println(\n=== STARTING MULTI-CUT L-SHAPED ITERATIONS ===)
max_iters = 50
UB = +Inf
LB = -Inf
for k in 1:max_iters
 println(\n----------------------------)
 println(Iteration $k)
 println(----------------------------)
 ########################
 # 1. Solve MASTER
```

```
#########################
optimize!(master)
if termination_status(master) != MOI.OPTIMAL
println(Master not optimal. Stopping.)
break
end
xk = value.(x)
k = value.()
investment_cost = sum(I[i] * xk[i] for i in 1:n)
LB = objective_value(master)
println([Master] x = , xk)
println([Master] Investment cost = , investment_cost)
println([Master]  per scenario = , k)
println([Master] LB = , LB)
#########################
# 2. Solve ALL SLAVES => scenario cuts
#########################
Q_vals = zeros()
Qbar = 0.0
# store per-scenario cut coefficients
 = zeros()
 = zeros(, n)
for  in 1:
Q, ,  = solve_slave_one_scenario(xk, , MC, T, V, D)
Q_vals[] = Q
Qbar += P[] * Q
[] = sum([j] * D[j, ] for j in 1:m)
for i in 1:n
[, i] = [i]
end
end
println([Slaves] Q = , Q_vals,  E[Q(x)] = , Qbar)
#########################
# 3. Update UB and gap
#########################
global UB
UB = min(UB, investment_cost + Qbar)
gap = UB - LB
println([Bounds] UB = , UB)
println([Bounds] Gap = , gap)
#########################
# 4. Store results row
#########################
row = Dict{Symbol, Any}()
row[:iter] = k
for i in 1:n
row[Symbol(names[i])] = xk[i]
end
for  in 1:
row[Symbol(_$())] = k[]
end
```

```
 row[:investment_cost] = investment_cost
 row[:EQ] = Qbar
 row[:LB] = LB
 row[:UB] = UB
 row[:gap] = gap
 row[:cut_type] = optimality_multi_cut
 push!(results, row)
 CSV.write(lshaped_multicut_results.csv, results)
 ##########################
 # 5. Convergence check
 ##########################
 if abs(gap) < 1e-3
 println(\n>>> CONVERGED <<<)
 break
 end
 ##########################
 # 6. Add NEW MULTI-CUTS: one per scenario
 ##########################
 for  in 1:
 cut = @constraint(master,
 [] >= [] + sum([, i] * x[i] for i in 1:n)
 )
 push!(cuts_by_scenario[], cut)
 end
 println(Added , ,  scenario optimality cuts (multi-cut).)
end
println(\n===== FINAL SOLUTION =====)
println(x* = , value.(x))
println(* per scenario = , value.())
println(\nResults saved to lshaped_multicut_results.csv.)
#################################################################
# Display from CSV
#################################################################
println(\n=== DISPLAYING OUTPUT FROM CSV ===)
df = CSV.read(lshaped_multicut_results.csv, DataFrame)
println(df)
```

Table 1: Καμπύλη διάρκειας φορτίου (οριζόντια διαστρωμάτωση).

Κατηγορία	Διάρκεια (ώρες)	Επίπεδο (MW)
Φορτίο Βάσης	8760	0–7086
Μέσο Φορτίο	7000	7086–9004
Φορτίο Αιχμής	1500	9004–11169

Table 2: Κόστη τεχνολογιών (λειτουργικό κόστος και κόστος αρχικής επένδυσης).

Τεχνολογία	Λειτουργικό κόστος (€/MWh)	Κόστος αρχικής επένδυσης (€/MWh)
Άνθρακας	25	16
Φυσικό Αέριο	80	5
Πυρηνικά	6.5	32
Πετρέλαιο	160	2

Table 4: Δεδομένα φορτίου για δύο σενάρια.

Κατηγορία	Διάρκεια (ώρες)	Σενάριο Αναφοράς (MW)	Σενάριο 10x Αιολικά (MW)
Φορτίο Βάσης	8760	0–7086	0–3919
Μέσο Φορτίο	7000	7086–9004	3919–7329
Φορτίο Αιχμής	1500	9004–11169	7329–10315