Optimizing Cyclone Shelter Allocation and Expansion in Teknaf Upazila

Contributors: Riya Parikh, Ayela Chughtai, Rose Dana

Imports - Data and Grid Loading

```
# Packages + Solver
using CSV, DataFrames, Statistics
using JuMP
using HiGHS
using CairoMakie

const SOLVER = HiGHS.Optimizer
```

```
HiGHS.Optimizer
```

```
# Load Data
grid = CSV.read("grid_pop.csv", DataFrame)
shelters_exist = CSV.read("shelter_final.csv", DataFrame)

# Column Mapping
# Grid cells (I)
col_grid_id   = :grid_id
col_latI      = :lat_center
col_lonI      = :lon_center
col_pop       = :pop_cell
col_risk      = :hazard_score   # risk uses hazard_score, not
hazard_class

# Existing shelters (J_E)
col_shel_id   = :CS_ID
col_latJE     = :lat
col_lonJE     = :lon
col_capJE     = Symbol("DRRO Capacity")
```

```
Symbol("DRRO Capacity")
```

Distance Calculations - Setting Candidate and Existing Shelters Onto Grids

```
# Helper Functions for Dist
deg2rad(x) = x * (pi / 180)

# Haversine distance measures between two lat/lon points in
kilometers.
function haversine_km(lat1, lon1, lat2, lon2)
```

```
    R = 6371.0  # Earth radius in km
    φ1 = deg2rad(lat1); φ2 = deg2rad(lat2)
    Δφ = deg2rad(lat2 - lat1)
    Δλ = deg2rad(lon2 - lon1)
    a = sin(Δφ/2)^2 + cos(φ1)*cos(φ2)*sin(Δλ/2)^2
    c = 2 * atan(sqrt(a), sqrt(1-a))
    return R * c
end
```

```
haversine_km (generic function with 1 method)
```

```
# Build Sets + Parameters
# Population cells (I)
I_ids = grid[!, col_grid_id]
latI  = grid[!, col_latI]
lonI  = grid[!, col_lonI]
d     = coalesce.(grid[!, col_pop], 0.0)      # population
r     = coalesce.(grid[!, col_risk], 0.0)     # risk score

nI = length(I_ids)

D_total = sum(d)
R_total = sum(r .* d)

# Existing shelters (J_E)
JE_ids = shelters_exist[!, col_shel_id]
latJE  = shelters_exist[!, col_latJE]
lonJE  = shelters_exist[!, col_lonJE]
uJE    = coalesce.(shelters_exist[!, col_capJE], 0.0)

nJE = length(JE_ids)
JE = 1:nJE

# Candidate shelters (J_N)
# = every grid center (NO filtering, duplicates allowed)
# New shelter capacity = max existing capacity
u_new = maximum(uJE)   # capacity ≤ max existing

JN_ids = ["cand_" * string(I_ids[i]) for i in 1:nI]
latJN  = latI
lonJN  = lonI
uJN    = fill(u_new, nI)

nJN = length(JN_ids)

# Combined shelter set J = J_E ∪ J_N
J_ids = vcat(JE_ids, JN_ids)
latJ  = vcat(latJE, latJN)
lonJ  = vcat(lonJE, lonJN)
u     = vcat(uJE,  uJN)
```

```
nJ = length(J_ids)

# candidate indices inside combined list
JN = (nJE+1):nJ
```

```
162:832
```

```
# Distance Matrix c_{ij}
c = Array{Float64}(undef, nI, nJ)
for i in 1:nI, j in 1:nJ
    c[i,j] = haversine_km(latI[i], lonI[i], latJ[j], lonJ[j])
end

# Hard Max Travel Distance Lmax (policy)
# Originally this was an extension, but model behaviour was odd if not
incorporated from the start
# St people were being assigned to shelters diagonally all the way
across the region
Lmax = 11.0   # in km

# M[i,j] = 1 if shelter j is within Lmax of cell i, else 0
M = Array{Int8}(undef, nI, nJ)
for i in 1:nI, j in 1:nJ
    M[i,j] = c[i,j] <= Lmax + 1e-9 ? 1 : 0
end
```

Solve 2 Stage Allocation + Location Problem

```
# Solver for One K
# Stage 1: maximize risk-weighted coverage
# Stage 1b: maximize use of existing shelters
# Stage 2: min distance with Lmax cutoff enforced in ALL stages

function solve_for_K(K::Int; delta_frac=1e-6)

    # Stage 1: Maximize TOTAL risk-weighted coverage S_r
    m1 = Model(SOLVER)
    set_silent(m1)

    @variable(m1, 0 <= x[1:nI, 1:nJ] <= 1)
    @variable(m1, y[1:nJ], Bin)

    for j in JE
        fix(y[j], 1.0; force=true)  # existing always open
    end

    @constraint(m1, [i=1:nI], sum(x[i,j] for j in 1:nJ) <= 1)
    @constraint(m1, [j=1:nJ], sum(d[i]*x[i,j] for i in 1:nI) <=
u[j]*y[j])
    @constraint(m1, sum(y[j] for j in JN) <= K)
```

```
    # Lmax cutoff
    @constraint(m1, [i=1:nI, j=1:nJ], x[i,j] <= M[i,j] * y[j])

    Sr_expr_m1 = @expression(m1, sum(r[i]*d[i]*x[i,j] for i in 1:nI, j
in 1:nJ))
    @objective(m1, Max, Sr_expr_m1)

    optimize!(m1)
    if termination_status(m1) != MOI.OPTIMAL
        error("Stage 1 failed for K=$K (might be infeasible with
Lmax=7km)")
    end

    Sr_star = objective_value(m1)
    δSr = delta_frac * R_total

    # Stage 1b: Among max-Sr solutions, maximize EXISTING use
    m1b = Model(SOLVER)
    set_silent(m1b)

    @variable(m1b, 0 <= xb[1:nI, 1:nJ] <= 1)
    @variable(m1b, yb[1:nJ], Bin)

    for j in JE
        fix(yb[j], 1.0; force=true)
    end

    @constraint(m1b, [i=1:nI], sum(xb[i,j] for j in 1:nJ) <= 1)
    @constraint(m1b, [j=1:nJ], sum(d[i]*xb[i,j] for i in 1:nI) <=
u[j]*yb[j])
    @constraint(m1b, sum(yb[j] for j in JN) <= K)

    # Lmax cutoff
    @constraint(m1b, [i=1:nI, j=1:nJ], xb[i,j] <= M[i,j] * yb[j])

    Sr_expr_m1b = @expression(m1b, sum(r[i]*d[i]*xb[i,j] for i in
1:nI, j in 1:nJ))
    @constraint(m1b, Sr_expr_m1b >= Sr_star - δSr)

    UE_expr = @expression(m1b, sum(d[i]*xb[i,j] for i in 1:nI, j in
JE))
    @objective(m1b, Max, UE_expr)

    optimize!(m1b)
    if termination_status(m1b) != MOI.OPTIMAL
        error("Stage 1b failed for K=$K")
    end

    UE_star = objective_value(m1b)
    δUE = delta_frac * D_total
```

```
    # Stage 2: Min distance among max-Sr AND max-UE solutions
    m2 = Model(SOLVER)
    set_silent(m2)

    @variable(m2, 0 <= x2[1:nI, 1:nJ] <= 1)
    @variable(m2, y2[1:nJ], Bin)

    for j in JE
        fix(y2[j], 1.0; force=true)
    end

    @constraint(m2, [i=1:nI], sum(x2[i,j] for j in 1:nJ) <= 1)
    @constraint(m2, [j=1:nJ], sum(d[i]*x2[i,j] for i in 1:nI) <=
u[j]*y2[j])
    @constraint(m2, sum(y2[j] for j in JN) <= K)

    # Lmax cutoff
    @constraint(m2, [i=1:nI, j=1:nJ], x2[i,j] <= M[i,j] * y2[j])

    Sr_expr_m2 = @expression(m2, sum(r[i]*d[i]*x2[i,j] for i in 1:nI,
j in 1:nJ))
    @constraint(m2, Sr_expr_m2 >= Sr_star - δSr)

    UE_expr_m2 = @expression(m2, sum(d[i]*x2[i,j] for i in 1:nI, j in
JE))
    @constraint(m2, UE_expr_m2 >= UE_star - δUE)

    @objective(m2, Min, sum(c[i,j]*d[i]*x2[i,j] for i in 1:nI, j in
1:nJ))

    optimize!(m2)
    if termination_status(m2) != MOI.OPTIMAL
        error("Stage 2 failed for K=$K")
    end

    x_sol = value.(x2)
    y_sol = value.(y2)

    Sr_realized = sum(r[i]*d[i]*x_sol[i,j] for i in 1:nI, j in 1:nJ)
    S_raw       = sum(d[i]*x_sol[i,j] for i in 1:nI, j in 1:nJ)
    UE_realized = sum(d[i]*x_sol[i,j] for i in 1:nI, j in JE)

    alpha_r = R_total > 0 ? Sr_realized / R_total : 0.0
    alpha   = D_total > 0 ? S_raw / D_total       : 0.0
    Z_star  = objective_value(m2)

    return (alpha_r=alpha_r, alpha=alpha, distance=Z_star,
            Sr_star=Sr_star, UE_star=UE_star, UE_realized=UE_realized,
```

```
            y=y_sol, x=x_sol)
end
```

```
solve_for_K (generic function with 1 method)
```

Understanding Hazard Classes

```
# Hazard-class prep (classes 1–4)
col_hazclass = :hazard_class
hclass = coalesce.(grid[!, col_hazclass], 0)

haz_classes = 1:4

# totals by class (raw pop and risk-mass)
total_pop_by_class = Dict(h => sum(d[i] for i in 1:nI if hclass[i] ==
h) for h in haz_classes)
total_riskmass_by_class = Dict(h => sum(r[i]*d[i] for i in 1:nI if
hclass[i] == h) for h in haz_classes)

println("Total pop by class: ", total_pop_by_class)
println("Total risk-mass by class: ", total_riskmass_by_class)

# Hazard-class coverage for a specific solution
# Compute hazard-class coverage for a given solution.
# Returns dicts:
# - raw_cov[h]  = covered pop in class h / total pop class h
# - risk_cov[h] = covered riskmass in class h / total riskmass class h

function hazard_class_coverage(sol)
    xK = sol.x
    # served fraction per cell s_i = sum_j x_ij
    s = [sum(xK[i,j] for j in 1:nJ) for i in 1:nI]

    raw_cov  = Dict{Int,Float64}()
    risk_cov = Dict{Int,Float64}()

    for h in haz_classes
        tot_pop_h  = total_pop_by_class[h]
        tot_risk_h = total_riskmass_by_class[h]

        covered_pop_h  = sum(d[i]*s[i] for i in 1:nI if hclass[i] ==
h)
        covered_risk_h = sum(r[i]*d[i]*s[i] for i in 1:nI if hclass[i]
== h)

        raw_cov[h]  = tot_pop_h  > 0 ? covered_pop_h  / tot_pop_h  :
0.0
        risk_cov[h] = tot_risk_h > 0 ? covered_risk_h / tot_risk_h :
0.0
    end
```

```
    return raw_cov, risk_cov
end
```

```
Total pop by class: Dict(4 => 123923, 2 => 122594, 3 => 142405, 1 =>
76142)
Total risk-mass by class: Dict(4 => 24668.602285310284, 2 =>
63791.00023895401, 3 => 45042.94259469208, 1 => 61522.72756027419)
```

```
hazard_class_coverage (generic function with 1 method)
```

```
# Frontier sweep over K (stores hazard-class metrics too)
# Run model for vector of K values and store frontier metrics +
hazard-class coverage.
# Returns:
# - frontier_df (wide, one row per K)
# - haz_long_df (long, one row per K per class)

function run_frontier(K_vals::Vector{Int})
    rows = NamedTuple[]
    haz_rows = NamedTuple[]
    total = length(K_vals)

    for (idx, K) in enumerate(K_vals)
        # progress update line
        pct = round(100 * idx / total; digits=1)
        println("\n[$idx/$total | $pct%] Solving K = $K")

        try
            solK = solve_for_K(K)
            open_new = sum(round.(Int, solK.y[j]) for j in JN)

            # hazard class coverage
            raw_cov, risk_cov = hazard_class_coverage(solK)

            push!(rows, (
                K=K,
                alpha_r=solK.alpha_r,
                alpha=solK.alpha,
                distance=solK.distance,
                open_new=open_new,
                status="optimal"
            ))

            for h in haz_classes
                push!(haz_rows, (
                    K=K,
                    hazard_class=h,
                    raw_cov=raw_cov[h],
                    risk_cov=risk_cov[h]
                ))
            end
```

```
        catch e
            println("K=$K failed: ", e)
            push!(rows, (
                K=K,
                alpha_r=missing,
                alpha=missing,
                distance=missing,
                open_new=missing,
                status="failed"
            ))
        end
    end

    frontier_df = DataFrame(rows)
    haz_long_df = DataFrame(haz_rows)
    return frontier_df, haz_long_df
end


# choosing K values to sweep
K_vals = [0, 5, 10, 15, 20, 25, 30]
frontier_df, haz_long_df = run_frontier(K_vals)

# save outputs
CSV.write("frontier_results.csv", frontier_df)
CSV.write("hazard_class_coverage.csv", haz_long_df)

println("Frontier results:")
show(frontier_df, allrows=true, allcols=true)
```

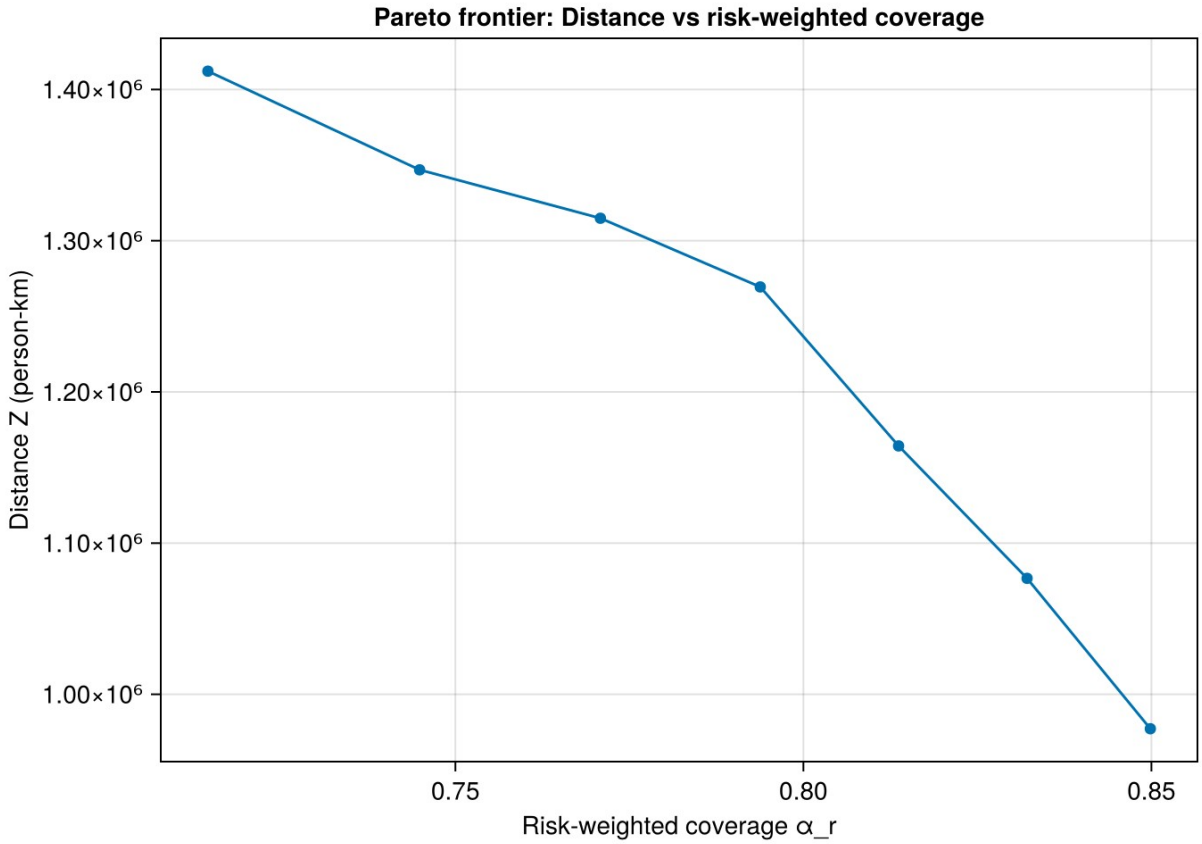
```
[1/7 | 14.3%] Solving K = 0

[2/7 | 28.6%] Solving K = 5

[3/7 | 42.9%] Solving K = 10

[4/7 | 57.1%] Solving K = 15

[5/7 | 71.4%] Solving K = 20

[6/7 | 85.7%] Solving K = 25

[7/7 | 100.0%] Solving K = 30
Frontier results:
7×6 DataFrame
 Row │ K      alpha_r   alpha     distance   open_new  status
     │ Int64  Float64   Float64   Float64    Int64     String
─────┼──────────────────────────────────────────────────────────
   1 │     0  0.71444   0.533976  1.41213e6         0  optimal
```

```
   2 │     5  0.744859  0.560854  1.34688e6         5  optimal
   3 │    10  0.770835  0.587732  1.31485e6        10  optimal
   4 │    15  0.793799  0.61461   1.26943e6        15  optimal
   5 │    20  0.813676  0.641488  1.16427e6        20  optimal
   6 │    25  0.832134  0.668366  1.0767e6         25  optimal
   7 │    30  0.849844  0.695244  9.77223e5        30  optimal
```

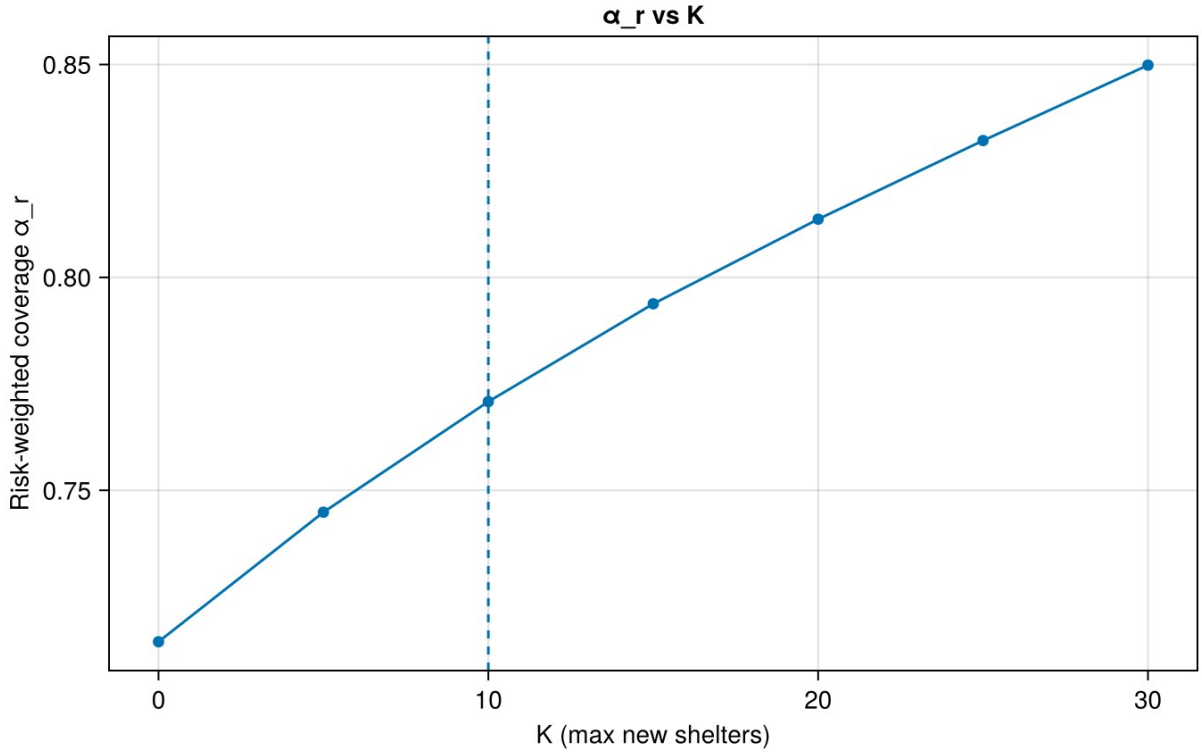
Plotting

```
# Elbow K* based on diminishing returns on alpha_r
# Choose K* by diminishing returns on alpha_r.
# eps_gain=0.002 means <0.2% gain per shelter counts as flat.
# window=3 means flat for 3 shelters in a row.

function pick_elbow_K(good_df; eps_gain=0.002, window=3)
    good_df = sort(good_df, :K)
    ar = good_df.alpha_r
    Ks = good_df.K

    d_ar = [ar[t] - ar[t-1] for t in 2:length(ar)]

    for t in 1:(length(d_ar) - window + 1)
        if all(d_ar[t:(t+window-1)] .< eps_gain)
            return Ks[t]
        end
    end
    return Ks[end]
end

good = frontier_df[frontier_df.status .== "optimal", :]
K_star = pick_elbow_K(good; eps_gain=0.002, window=3)

# Looking back from our plots, actually, we believe that K=10 (open 10
new shelters) is sufficient.
# This allows us to cover all of group 1 and 2 (highest risk groups),
while still covering over 50% of risk group 3.
# Risk group 4 is almost never affacted by cyclones due to their
positioning releative to the sea and their elevation,
# so they are not our priority.

K_star = 10

println("\nElbow-selected K* = ", K_star)


Elbow-selected K* = 10

# Frontier plots + Pareto curves
good = sort(good, :K)

# alpha_r vs K
```

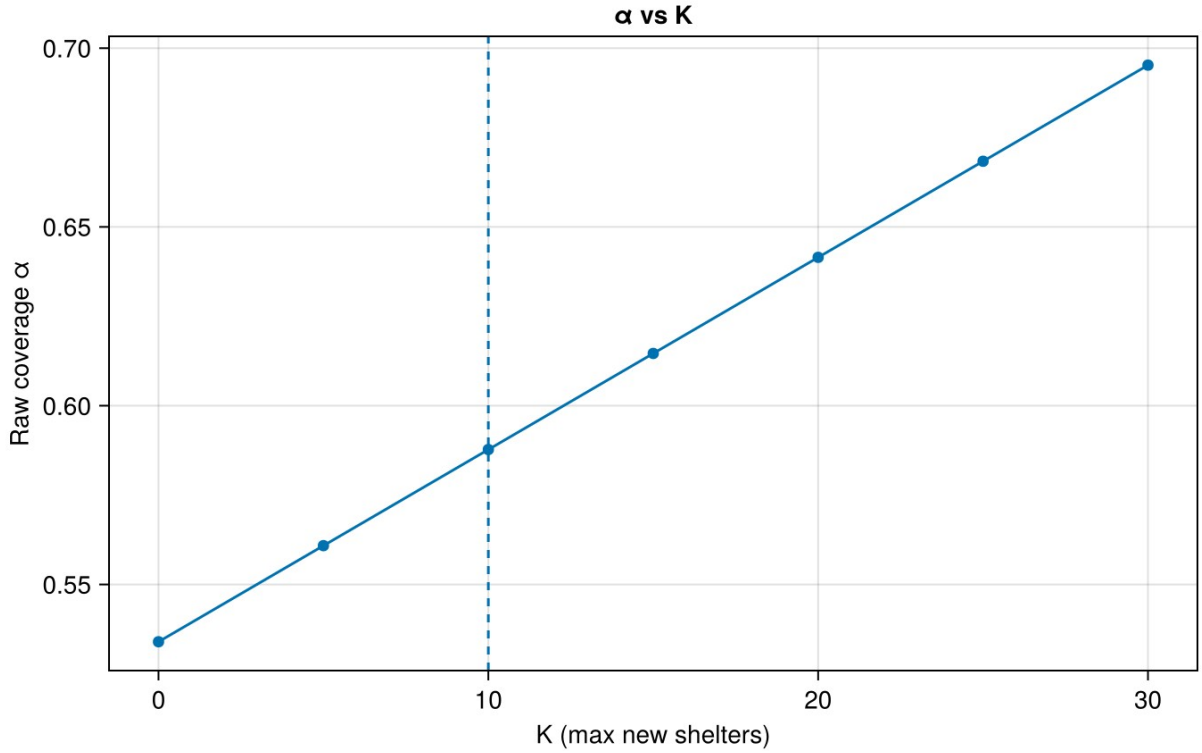
```
fig1 = Figure(size=(700,450))
ax1 = Axis(fig1[1,1], xlabel="K (max new shelters)", ylabel="Risk-
weighted coverage α_r",
           title="α_r vs K")
lines!(ax1, good.K, good.alpha_r)
scatter!(ax1, good.K, good.alpha_r)
vlines!(ax1, [K_star], linestyle=:dash)
fig1

# alpha vs K
fig2 = Figure(size=(700,450))
ax2 = Axis(fig2[1,1], xlabel="K (max new shelters)", ylabel="Raw
coverage α",
           title="α vs K")
lines!(ax2, good.K, good.alpha)
scatter!(ax2, good.K, good.alpha)
vlines!(ax2, [K_star], linestyle=:dash)
fig2

# distance vs K
fig3 = Figure(size=(700,450))
ax3 = Axis(fig3[1,1], xlabel="K (max new shelters)", ylabel="Total
distance Z (person-km)",
           title="Distance objective vs K")
lines!(ax3, good.K, good.distance)
scatter!(ax3, good.K, good.distance)
vlines!(ax3, [K_star], linestyle=:dash)
fig3

# Pareto: distance vs raw alpha
fig4 = Figure(size=(700,500))
ax4 = Axis(fig4[1,1], xlabel="Raw coverage α", ylabel="Distance Z
(person-km)",
           title="Pareto frontier: Distance vs raw coverage")
lines!(ax4, good.alpha, good.distance)
scatter!(ax4, good.alpha, good.distance)
fig4

# Pareto: distance vs alpha_r
fig5 = Figure(size=(700,500))
ax5 = Axis(fig5[1,1], xlabel="Risk-weighted coverage α_r",
ylabel="Distance Z (person-km)",
           title="Pareto frontier: Distance vs risk-weighted
coverage")
lines!(ax5, good.alpha_r, good.distance)
scatter!(ax5, good.alpha_r, good.distance)
fig5
```


Pareto frontier: Distance vs risk-weighted coverage
1.40×10⁶
1.30×10⁶
1.20×10⁶
1.10×10⁶
1.00×10⁶
Distance Z (person-km)
0.75
0.80
0.85
Risk-weighted coverage α_r

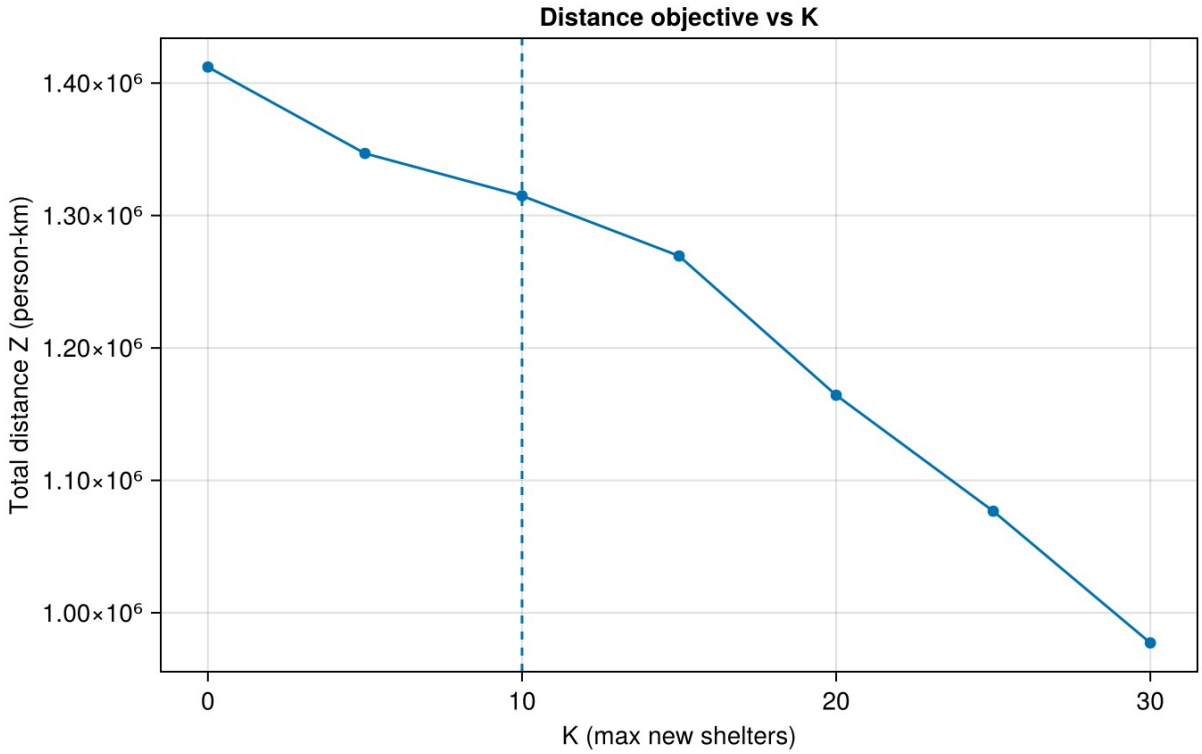
```
fig1
```


α_r vs K
Risk-weighted coverage α_r
0.85
0.80
0.75
0
10
20
30
K (max new shelters)

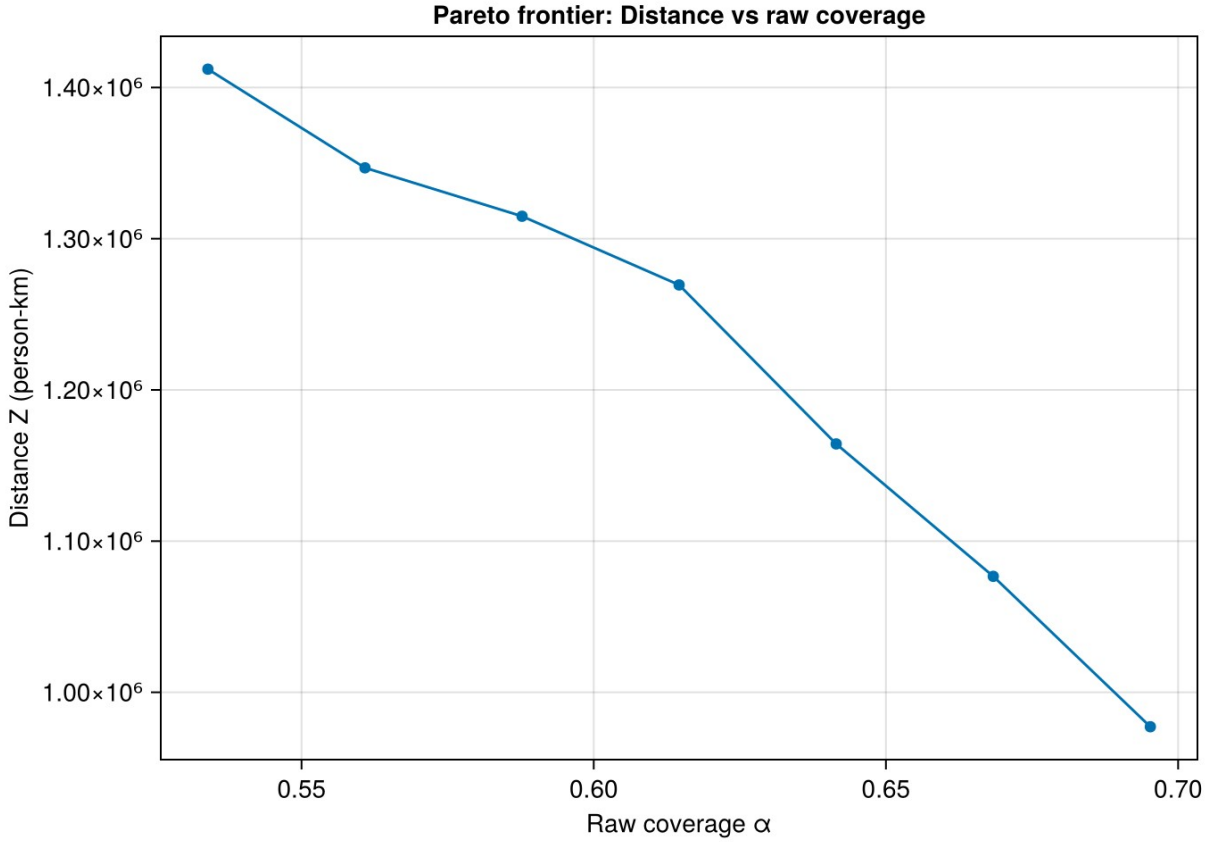
```
fig2
```


α vs K
Raw coverage α
0.70
0.65
0.60
0.55
0
10
20
30
K (max new shelters)

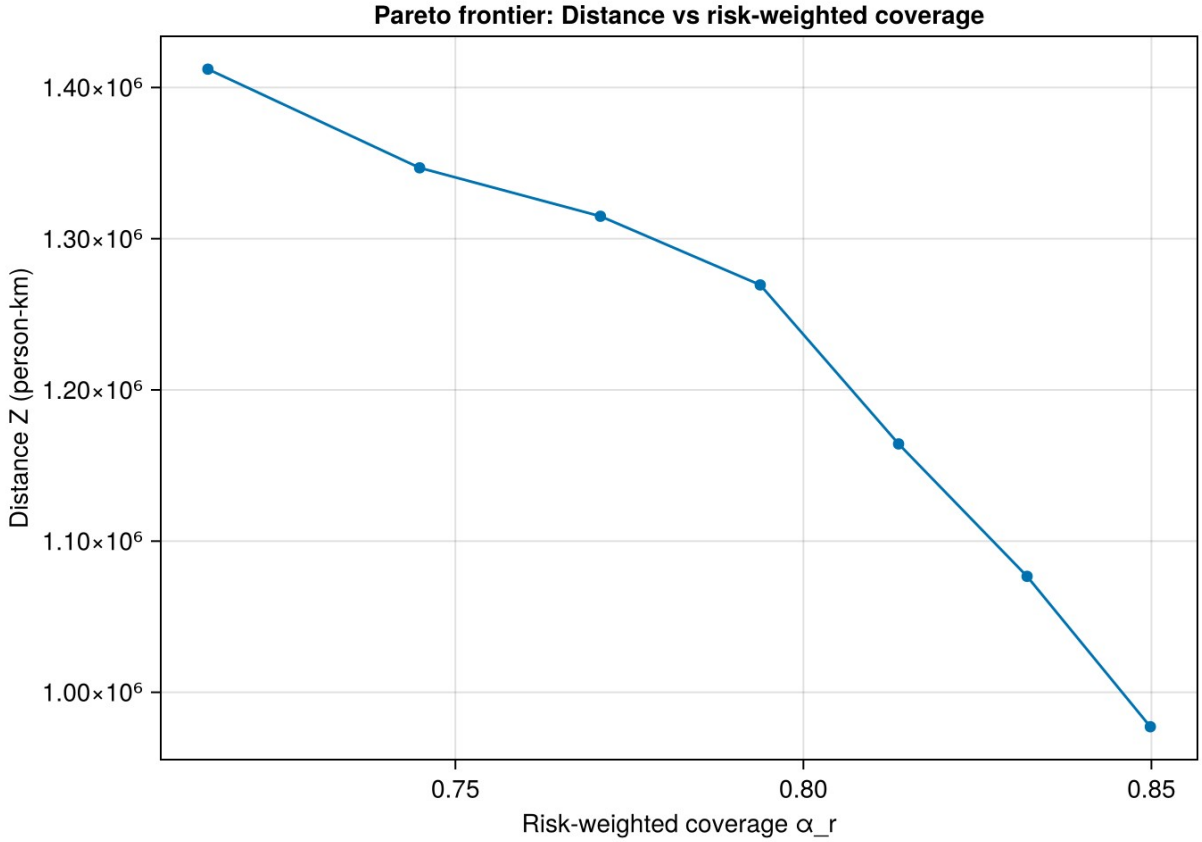
```
fig3
```


Distance objective vs K
1.40×10⁶
1.30×10⁶
1.20×10⁶
1.10×10⁶
1.00×10⁶
Total distance Z (person-km)
0
10
20
30
K (max new shelters)

```
fig4
```

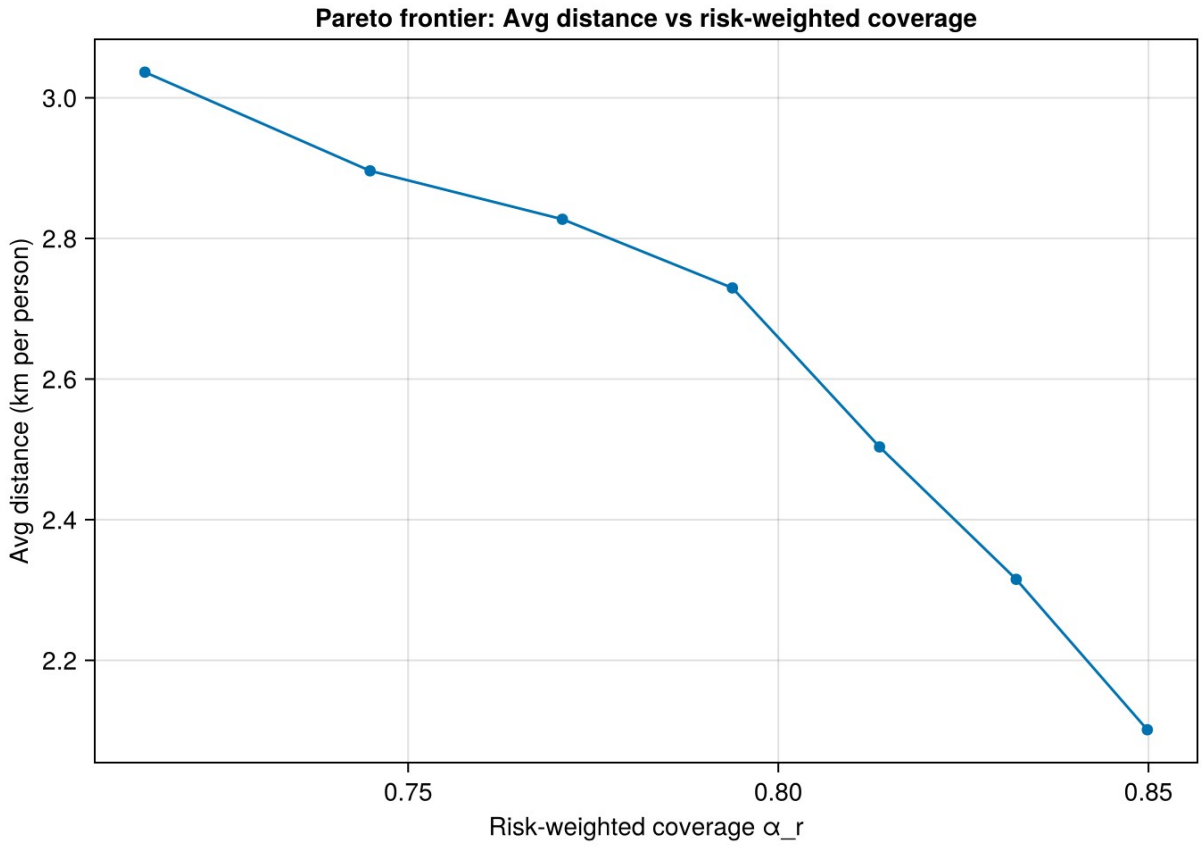

Pareto frontier: Distance vs raw coverage
Distance Z (person-km)
1.40×10⁶
1.30×10⁶
1.20×10⁶
1.10×10⁶
1.00×10⁶
0.55
0.60
0.65
0.70
Raw coverage α

```
fig5
```

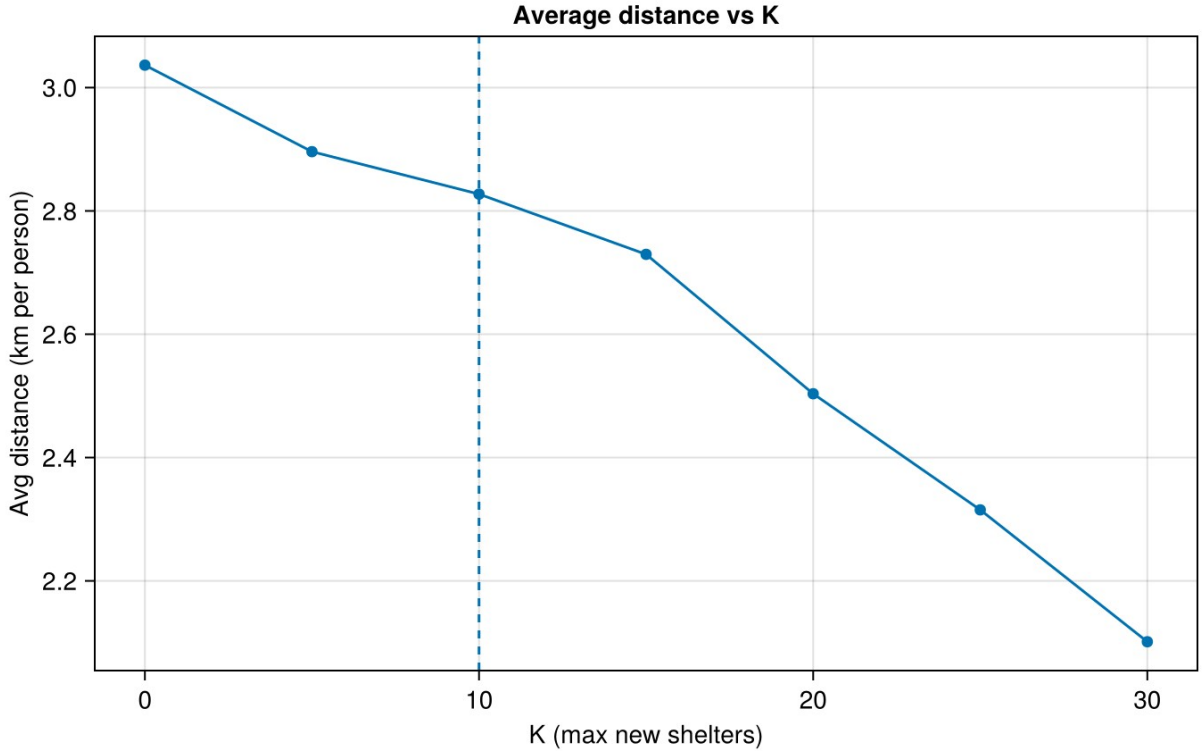



```
good = sort(good, :K)

total_pop_by_class = Dict(4 => 123923, 2 => 122594, 3 => 142405, 1 =>
76142)
denom = sum(values(total_pop_by_class))   # = 465_064

good.avg_distance = good.distance ./ denom

# avg distance vs K  (fig3a)
fig3a = Figure(size=(700,450))
ax3a = Axis(fig3a[1,1],
    xlabel="K (max new shelters)",
    ylabel="Avg distance (km per person)",
    title="Average distance vs K"
)
lines!(ax3a, good.K, good.avg_distance)
scatter!(ax3a, good.K, good.avg_distance)
vlines!(ax3a, [K_star], linestyle=:dash)
fig3a

# Pareto: avg distance vs raw alpha (fig4a)
fig4a = Figure(size=(700,500))
ax4a = Axis(fig4a[1,1],
```

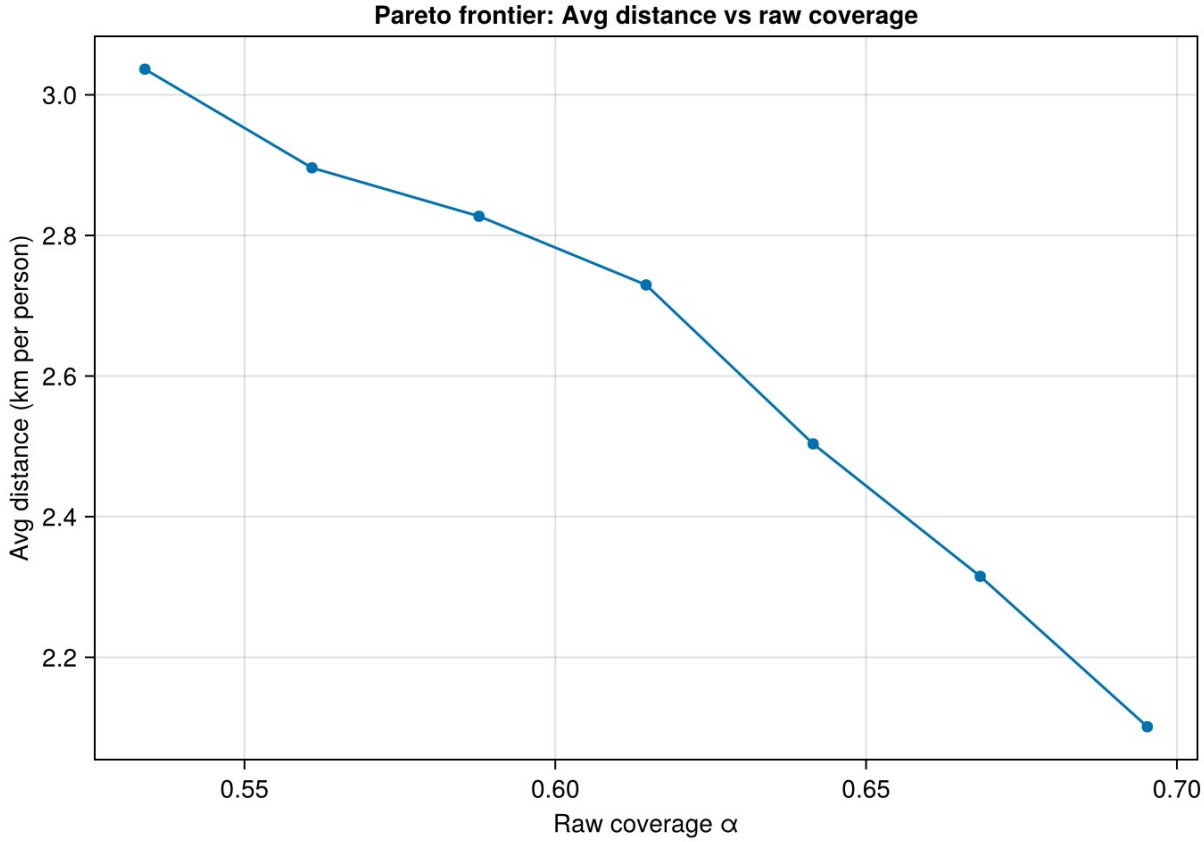
```
    xlabel="Raw coverage α",
    ylabel="Avg distance (km per person)",
    title="Pareto frontier: Avg distance vs raw coverage"
)
lines!(ax4a, good.alpha, good.avg_distance)
scatter!(ax4a, good.alpha, good.avg_distance)
fig4a

# Pareto: avg distance vs alpha_r (fig5a)
fig5a = Figure(size=(700,500))
ax5a = Axis(fig5a[1,1],
    xlabel="Risk-weighted coverage α_r",
    ylabel="Avg distance (km per person)",
    title="Pareto frontier: Avg distance vs risk-weighted coverage"
)
lines!(ax5a, good.alpha_r, good.avg_distance)
scatter!(ax5a, good.alpha_r, good.avg_distance)
fig5a
```



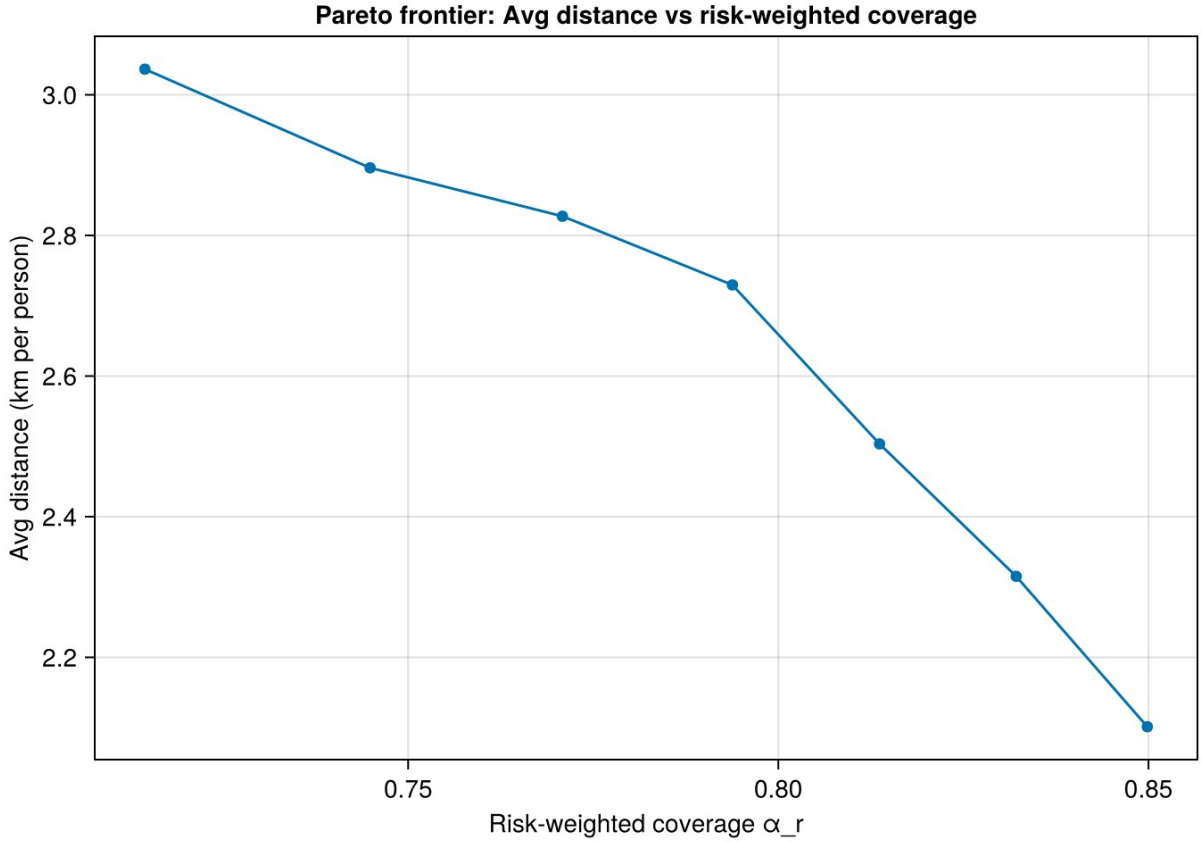

```
fig3a
```


Average distance vs K
Avg distance (km per person)
3.0
2.8
2.6
2.4
2.2
0
10
20
30
K (max new shelters)

```
fig4a
```

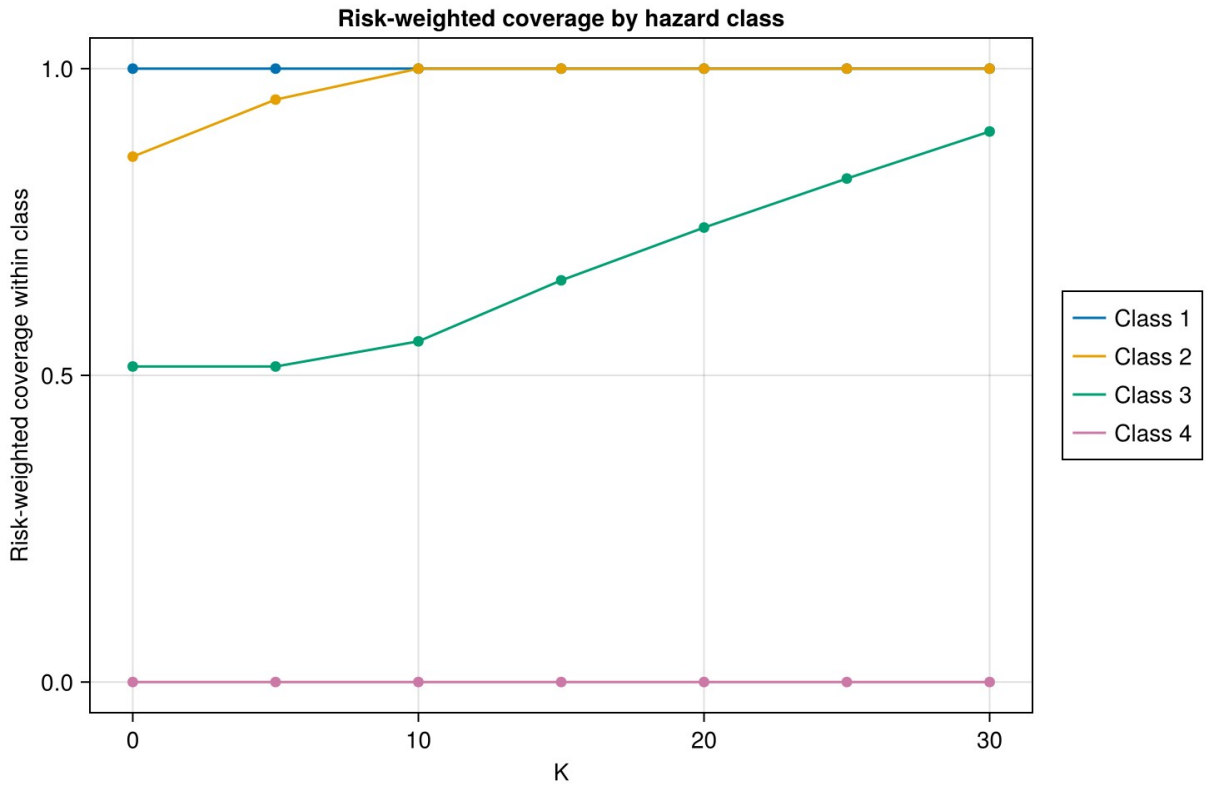

Pareto frontier: Avg distance vs raw coverage
Avg distance (km per person)
3.0
2.8
2.6
2.4
2.2
0.55
0.60
0.65
0.70
Raw coverage α

```
fig5a
```

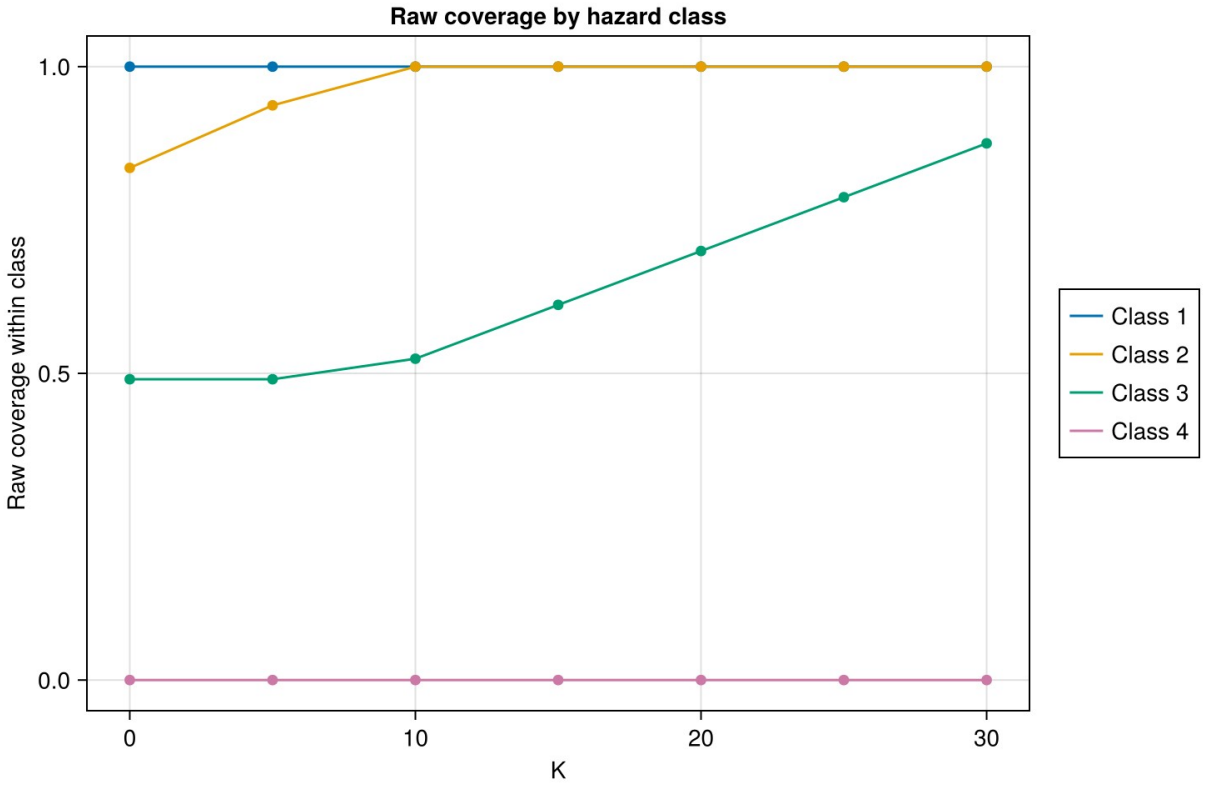



```
# Hazard-class coverage plots (raw + risk) vs K
haz_good = haz_long_df[in.(haz_long_df.K, Ref(good.K)), :]

# raw coverage by class vs K
fig6 = Figure(size=(750,500))
ax6 = Axis(fig6[1,1], xlabel="K", ylabel="Raw coverage within class",
           title="Raw coverage by hazard class")
for h in haz_classes
    sub = haz_good[haz_good.hazard_class .== h, :]
    lines!(ax6, sub.K, sub.raw_cov, label="Class $h")
    scatter!(ax6, sub.K, sub.raw_cov)
end
Legend(fig6[1,2], ax6)
fig6

# risk-weighted coverage by class vs K
fig7 = Figure(size=(750,500))
ax7 = Axis(fig7[1,1], xlabel="K", ylabel="Risk-weighted coverage
within class",
           title="Risk-weighted coverage by hazard class")
for h in haz_classes
    sub = haz_good[haz_good.hazard_class .== h, :]
    lines!(ax7, sub.K, sub.risk_cov, label="Class $h")
```

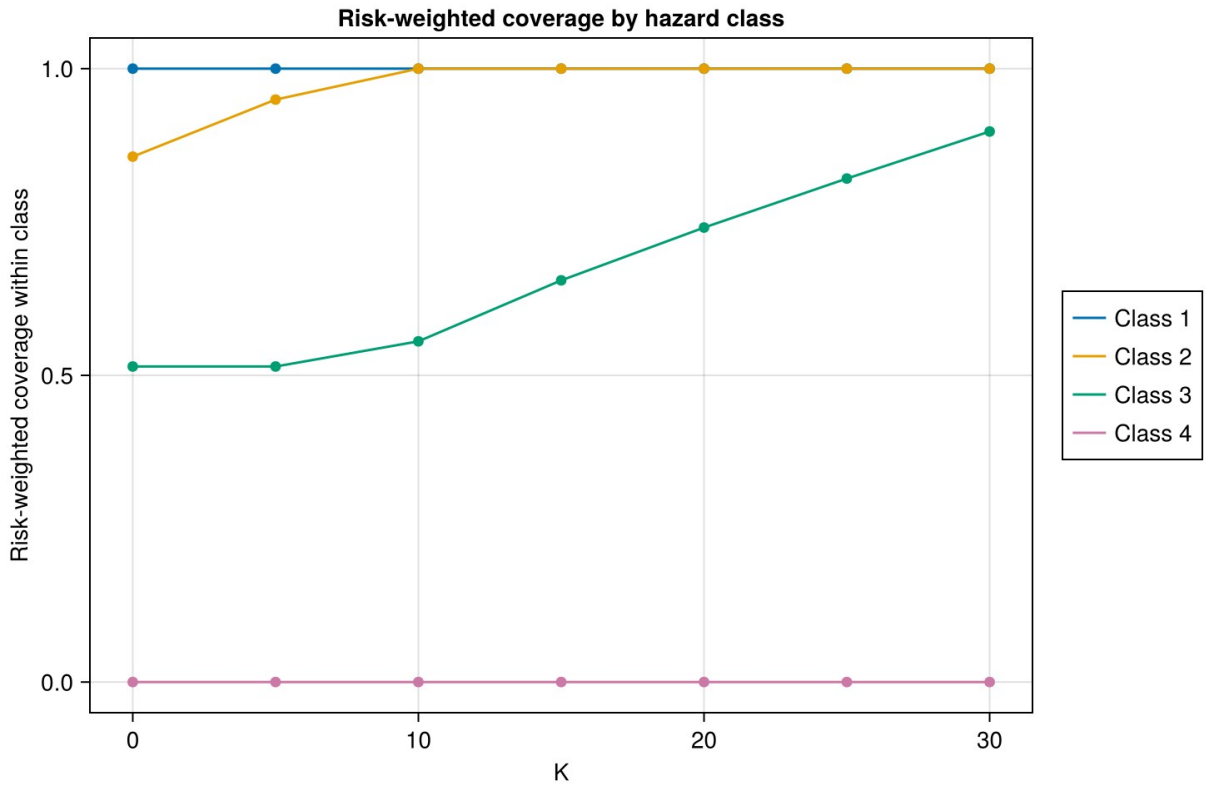
```
    scatter!(ax7, sub.K, sub.risk_cov)
end
Legend(fig7[1,2], ax7)
fig7
```




```
fig6
```


Raw coverage by hazard class
Raw coverage within class
1.0
0.5
0.0
0
10
20
30
K
Class 1
Class 2
Class 3
Class 4

```
fig7
```




```julia
# Print hazard-class coverage table for a specific K.
function show_class_coverage(K::Int)
    sub = haz_long_df[haz_long_df.K .== K, :]
    println("Hazard-class coverage for K=$K")
    show(sub, allrows=true, allcols=true)
    return sub
end

# Use like this:
# show_class_coverage(0)
# show_class_coverage(K_star)
```

```
show_class_coverage (generic function with 1 method)
```

```julia
show_class_coverage(0)
```

```
Hazard-class coverage for K=0
4×4 DataFrame
 Row │ K      hazard_class  raw_cov   risk_cov
     │ Int64  Int64         Float64   Float64
─────┼──────────────────────────────────────────
   1 │     0             1  1.0       1.0
   2 │     0             2  0.834935  0.856552
   3 │     0             3  0.490383  0.514415
   4 │     0             4  0.0       0.0
```

```
4×4 DataFrame
 Row │ K      hazard_class  raw_cov   risk_cov
     │ Int64  Int64         Float64   Float64
─────┼──────────────────────────────────────────
   1 │     0             1  1.0       1.0
   2 │     0             2  0.834935  0.856552
   3 │     0             3  0.490383  0.514415
   4 │     0             4  0.0       0.0
```

```
show_class_coverage(K_star)
```

```
Hazard-class coverage for K=10
4×4 DataFrame
 Row │ K      hazard_class  raw_cov   risk_cov
     │ Int64  Int64         Float64   Float64
─────┼──────────────────────────────────────────
   1 │    10             1  1.0       1.0
   2 │    10             2  1.0       1.0
   3 │    10             3  0.523837  0.555436
   4 │    10             4  0.0       0.0

4×4 DataFrame
 Row │ K      hazard_class  raw_cov   risk_cov
     │ Int64  Int64         Float64   Float64
─────┼──────────────────────────────────────────
   1 │    10             1  1.0       1.0
   2 │    10             2  1.0       1.0
   3 │    10             3  0.523837  0.555436
   4 │    10             4  0.0       0.0
```

Baseline (K=0) vs Elbow-best Maps - these are most useful for comparison!

```
# Baseline (K=0) vs elbow-best maps
function plot_coverage_maps(sol; title_prefix="")
    # served fraction s_i
    s = [sum(sol.x[i,j] for j in 1:nJ) for i in 1:nI]

    raw_color = s
    risk_color = r .* s
    risk_color_norm = maximum(risk_color) > 0 ? risk_color ./
maximum(risk_color) : risk_color

    open_mask = round.(Int, sol.y) .== 1
    new_mask  = [j in JN for j in 1:nJ]
    open_existing = open_mask .& .!new_mask
    open_new      = open_mask .& new_mask

    fig = Figure(size=(1200,520))

    ax1 = Axis(fig[1,1], aspect=DataAspect(),
        title=title_prefix * "Raw coverage (served fraction)",
```

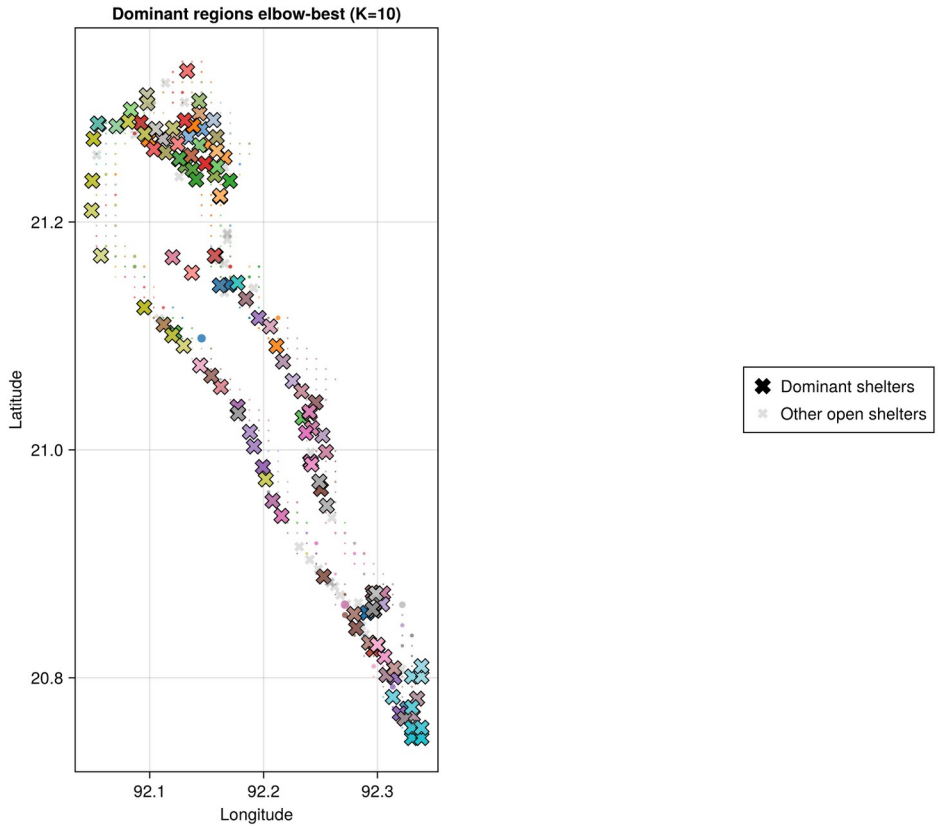
```
        xlabel="Longitude", ylabel="Latitude")

    sc1 = scatter!(ax1, lonI, latI;
        color=raw_color, colormap=:viridis, markersize=5, alpha=0.95)

    scatter!(ax1, lonJ[open_existing], latJ[open_existing];
        marker=:x, markersize=14, color=:red, label="Existing
shelters")
    scatter!(ax1, lonJ[open_new], latJ[open_new];
        marker=:x, markersize=14, color=:teal, label="New shelters")

    Colorbar(fig[1,2], sc1)   # pass plot handle, not axis


    ax2 = Axis(fig[1,3], aspect=DataAspect(),
        title=title_prefix * "Risk-weighted coverage intensity",
        xlabel="Longitude", ylabel="Latitude")

    sc2 = scatter!(ax2, lonI, latI;
        color=risk_color_norm, colormap=:magma, markersize=5,
alpha=0.95)

    scatter!(ax2, lonJ[open_existing], latJ[open_existing];
        marker=:x, markersize=14, color=:red, label="Existing
shelters")
    scatter!(ax2, lonJ[open_new], latJ[open_new];
        marker=:x, markersize=14, color=:teal, label="New shelters")

    Colorbar(fig[1,4], sc2)
    Legend(fig[1,5], ax2)     # outside legend

    fig
end

function plot_assignments(sol; title="", flow_thresh=100)
    rows = NamedTuple[]
    for i in 1:nI, j in 1:nJ
        if sol.x[i,j] > 1e-6
            push!(rows, (i=i, j=j, pop_assigned=d[i]*sol.x[i,j]))
        end
    end
    assign_df = DataFrame(rows)
    assign_plot = assign_df[assign_df.pop_assigned .>= flow_thresh, :]

    open_mask = round.(Int, sol.y) .== 1
    new_mask  = [j in JN for j in 1:nJ]
    open_existing = open_mask .& .!new_mask
    open_new      = open_mask .& new_mask

    fig = Figure(size=(1050,720))
```

```
    ax  = Axis(fig[1,1], aspect=DataAspect(),
               xlabel="Longitude", ylabel="Latitude",
               title=title)

    for row in eachrow(assign_plot)
        i=row.i; j=row.j
        lines!(ax, [lonI[i], lonJ[j]], [latI[i], latJ[j]];
            linewidth=0.6, color=(:black, 0.12))
    end

    scatter!(ax, lonI, latI;
        markersize = 2 .+ 8 .* (d ./ maximum(d)),
        color = r, colormap=:viridis, alpha=0.75,
        label="Grid cells (risk color)"
    )

    scatter!(ax, lonJ[open_existing], latJ[open_existing];
        marker=:x, markersize=14, color=:red, label="Existing
shelters")
    scatter!(ax, lonJ[open_new], latJ[open_new];
        marker=:x, markersize=14, color=:teal, label="New shelters")

    Legend(fig[1,2], ax)
    fig
end

function plot_dominant_map(sol; title="", return_tables=false,
show_nondominant_open=true)
    # build dominant assignment per region i
    dom_rows = NamedTuple[]
    for i in 1:nI
        assigned_pop_i = sum(d[i] * sol.x[i,j] for j in 1:nJ)
        if assigned_pop_i <= 1e-6
            continue
        end
        j_dom = argmax(sol.x[i, :])
        push!(dom_rows, (i=i, lon=lonI[i], lat=latI[i], pop=d[i],
dom_j=j_dom))
    end
    dom_df = DataFrame(dom_rows)

    # unique dominant shelters and a stable ordering
    uniq_shelters = sort(unique(dom_df.dom_j))
    group_id = Dict(j => k for (k,j) in enumerate(uniq_shelters))
    dom_df.group = [group_id[j] for j in dom_df.dom_j]

    G = length(uniq_shelters)
    colors = cgrad(:tab20, G; categorical=true)  # discrete palette
    shel_group = collect(1:G)                    # 1..G for coloring
X's
```

```
    # masks for open / new / existing
    open_mask = round.(Int, sol.y) .== 1
    new_mask  = [j in JN for j in 1:nJ]
    open_existing = open_mask .& .!new_mask
    open_new      = open_mask .& new_mask

    dom_set = Set(uniq_shelters)
    other_open_inds = show_nondominant_open ?
        [j for j in 1:nJ if open_mask[j] && !(j in dom_set)] : Int[]

    # plot
    fig = Figure(size=(1050,720))
    ax  = Axis(fig[1,1], aspect=DataAspect(),
               xlabel="Longitude", ylabel="Latitude",
               title=title)

    # regions colored by dominant shelter  (no legend needed for this)
    scatter!(ax, dom_df.lon, dom_df.lat;
        color=dom_df.group, colormap=colors,
        markersize=2 .+ 8 .* (dom_df.pop ./ maximum(dom_df.pop)),
        alpha=0.9)

    # dominant shelters as X's colored by their group
    scatter!(ax, lonJ[uniq_shelters], latJ[uniq_shelters];
        color=shel_group, colormap=colors,
        marker=:x, markersize=18, strokecolor=:black, strokewidth=0.8,
        label="Dominant shelters")

    # show other open shelters faintly (still X)
    if show_nondominant_open && !isempty(other_open_inds)
        scatter!(ax, lonJ[other_open_inds], latJ[other_open_inds];
            marker=:x, markersize=12, color=(:gray, 0.25),
            label="Other open shelters")
    end

    Legend(fig[1,2], ax)

    # tables to tell "which shelter dominates where"
    # summary: how many cells + total dominated pop per shelter
    shelter_summary = combine(groupby(dom_df, :dom_j),
        nrow => :n_cells,
        :pop => sum => :dominated_pop
    )
    shelter_summary.lon = lonJ[shelter_summary.dom_j]
    shelter_summary.lat = latJ[shelter_summary.dom_j]
    shelter_summary.is_new = [j in JN for j in shelter_summary.dom_j]
    sort!(shelter_summary, :dominated_pop, rev=true)

    if return_tables
```

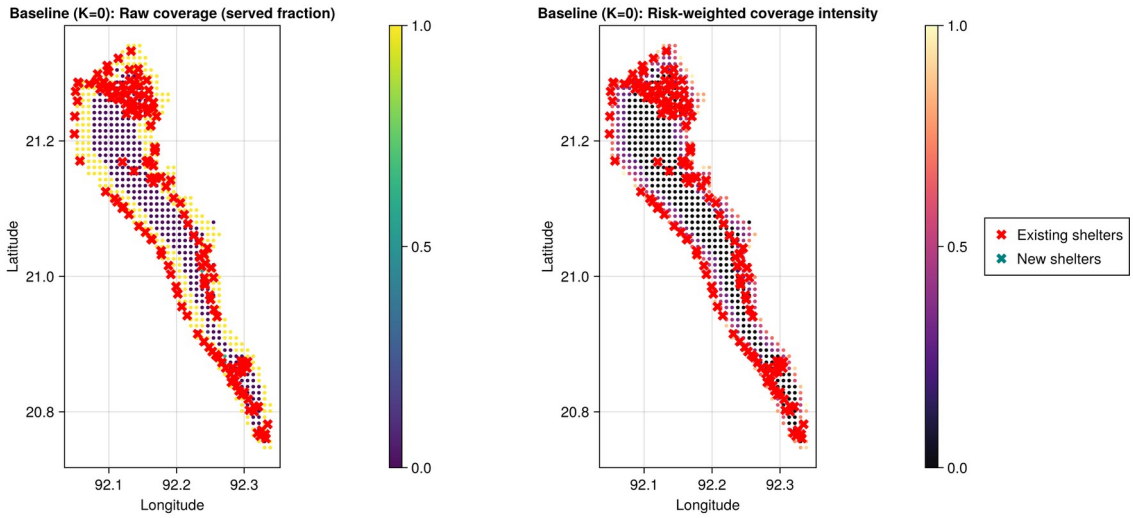
```
        return fig, dom_df, shelter_summary
    else
        return fig
    end
end

# solve baseline + elbow best - alr ran once dont need to again!
# sol_base  = solve_for_K(0)
# sol_star  = solve_for_K(K_star)

println("\nBaseline K=0: alpha_r=", sol_base.alpha_r,
        " alpha=", sol_base.alpha, " Z=", sol_base.distance)
println("Elbow best K=$K_star: alpha_r=", sol_star.alpha_r,
        " alpha=", sol_star.alpha, " Z=", sol_star.distance)

plot_coverage_maps(sol_base; title_prefix="Baseline (K=0): ")
plot_assignments(sol_base; title="Assignments baseline (K=0)")
plot_dominant_map(sol_base; title="Dominant regions baseline (K=0)")

plot_coverage_maps(sol_star; title_prefix="Elbow best (K=$K_star): ")
plot_assignments(sol_star; title="Assignments elbow-best (K=$K_star)")
plot_dominant_map(sol_star; title="Dominant regions elbow-best
(K=$K_star)")
```

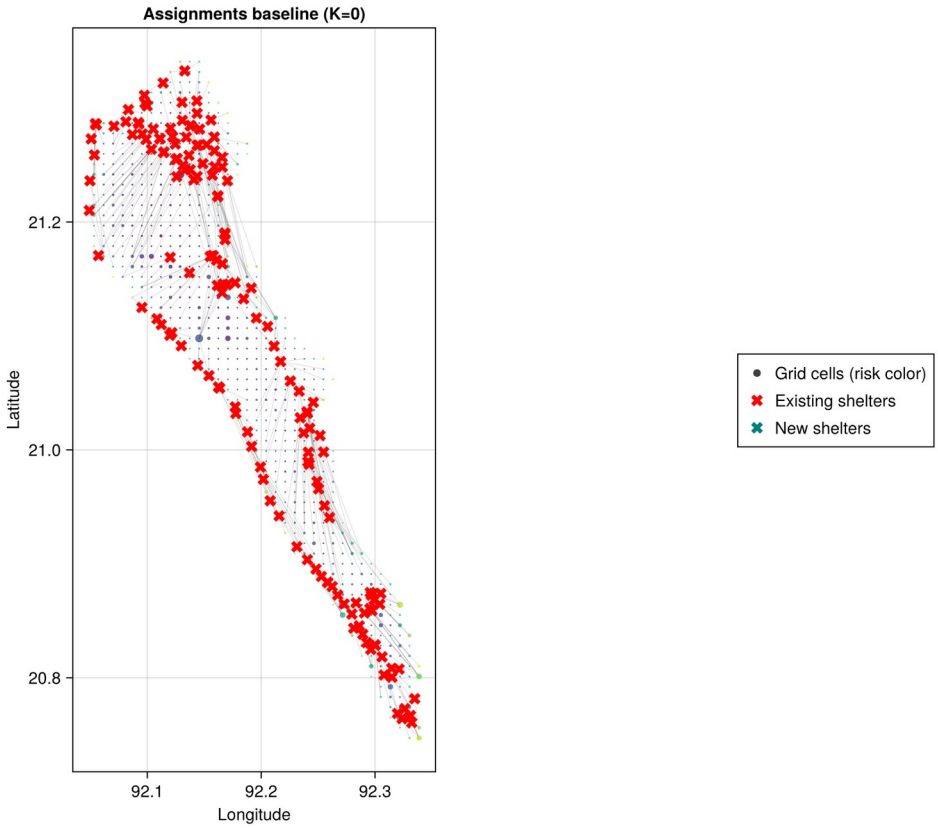
```
Baseline K=0: alpha_r=0.7144398315077909 alpha=0.5339759688989042
Z=1.4121322517052 55e6
Elbow best K=10: alpha_r=0.7708345652865948 alpha=0.5877320110780452
Z=1.3148536642687463e6
```

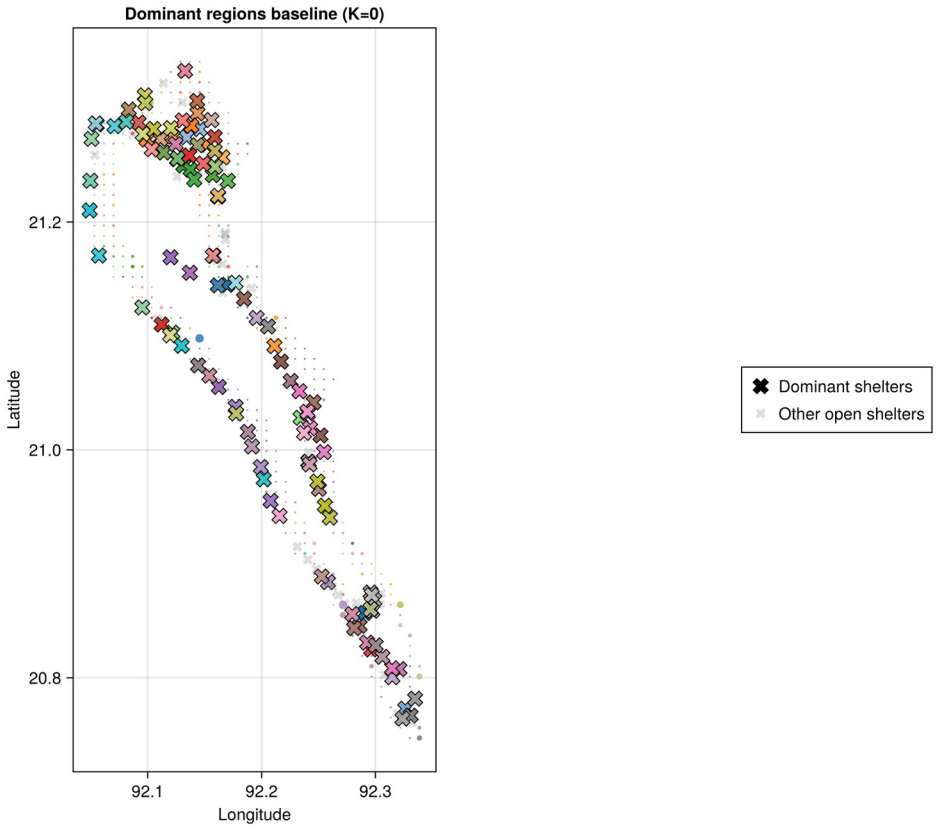


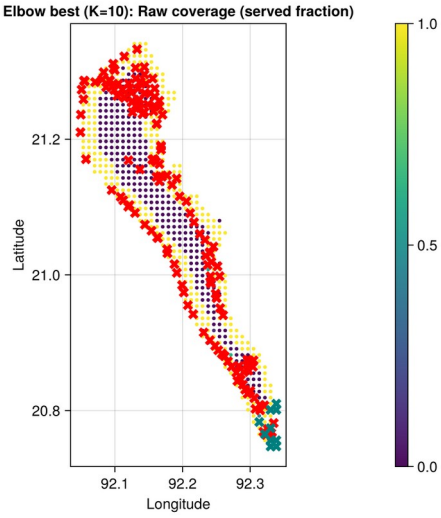

```
# collect figs for display
figs_cad = Figure[]

# Baseline figs (K=0)
push!(figs_cad, plot_coverage_maps(sol_base; title_prefix="Baseline
(K=0): "))
push!(figs_cad, plot_assignments(sol_base; title="Assignments baseline
(K=0)"))

# dominant baseline map (figure only)
fig_dom_base = plot_dominant_map(sol_base; title="Dominant regions
baseline (K=0)")
push!(figs_cad, fig_dom_base)


# Elbow-best / K* figs
push!(figs_cad, plot_coverage_maps(sol_star; title_prefix="Elbow best
(K=$K_star): "))
push!(figs_cad, plot_assignments(sol_star; title="Assignments elbow-
best (K=$K_star)"))

# dominant K* map + tables
fig_dom_star, dom_df_star, shelter_summary_star = plot_dominant_map(
    sol_star;
```

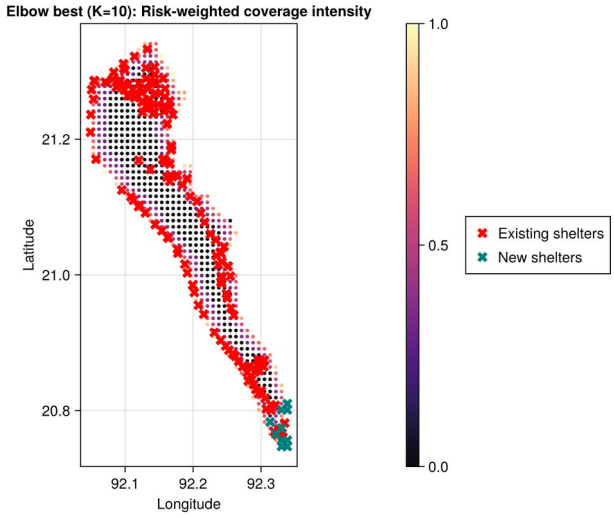
```
    title="Dominant regions at K*=$K_star",
    return_tables=true
)
push!(figs_cad, fig_dom_star)
foreach(display, figs_cad)

# print the dominance tables for K*
println("\nDominant-region rows (each grid cell i -> dominant shelter
dom_j):")
show(dom_df_star, allrows=true, allcols=true)

println("\nSummary by dominant shelter:")
show(shelter_summary_star, allrows=true, allcols=true)
```

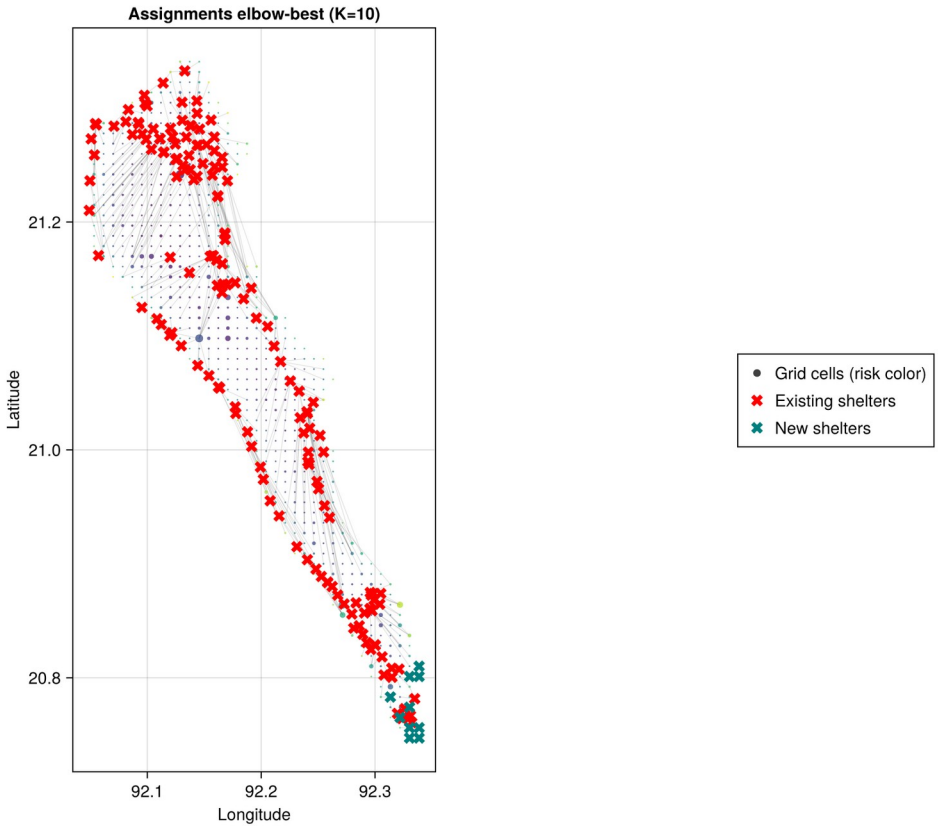


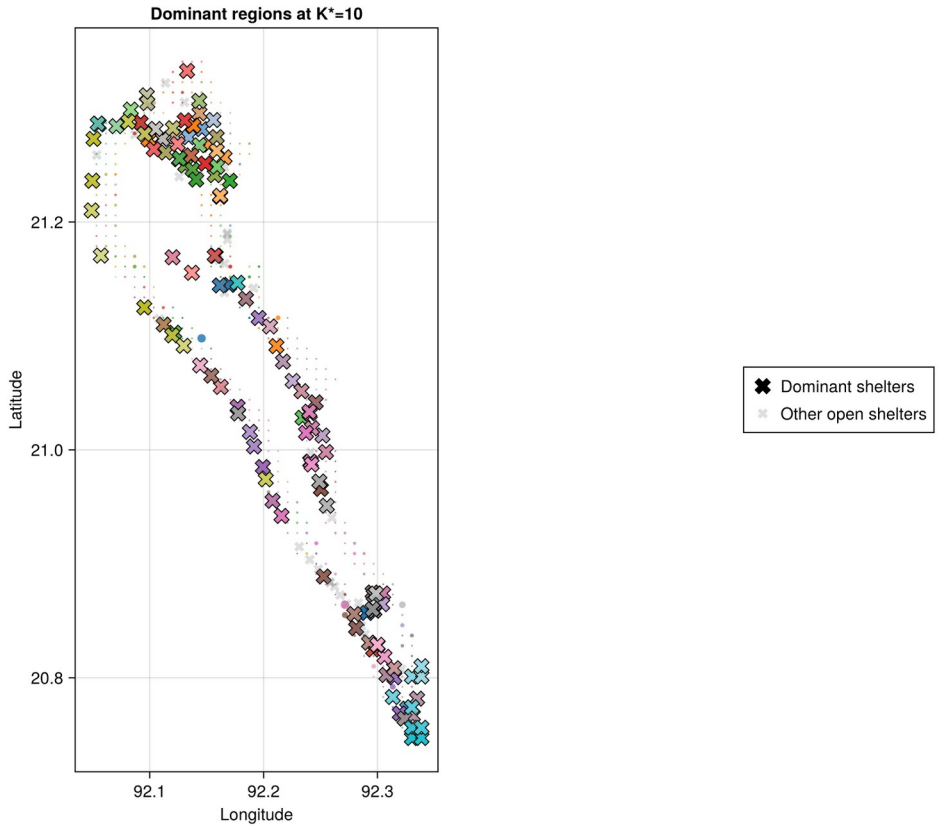

Assignments baseline (K=0)
Latitude
21.2
21.0
20.8
92.1
92.2
92.3
Longitude
Grid cells (risk color)
Existing shelters
New shelters


Dominant regions baseline (K=0)
Latitude
Longitude
21.2
21.0
20.8
92.1
92.2
92.3
Dominant shelters
Other open shelters


Elbow best (K=10): Raw coverage (served fraction)
Latitude
Longitude
21.2
21.0
20.8
92.1
92.2
92.3
1.0
0.5
0.0


Elbow best (K=10): Risk-weighted coverage intensity
Latitude
Longitude
21.2
21.0
20.8
92.1
92.2
92.3
1.0
0.5
0.0
Existing shelters
New shelters


Assignments elbow-best (K=10)
Latitude
21.2
21.0
20.8
92.1
92.2
92.3
Longitude
Grid cells (risk color)
Existing shelters
New shelters



Dominant-region rows (each grid cell i -> dominant shelter dom_j):
381×6 DataFrame

Row	i	lon	lat	pop	dom_j	group
	Int64	Float64	Float64	Int64	Int64	Int64
1	1	92.3303	20.7472	1211	162	118
2	2	92.3387	20.7472	5689	163	119
3	3	92.3219	20.7562	210	165	120
4	4	92.3303	20.7562	29	165	120
5	5	92.3387	20.7562	3244	166	121
6	6	92.3135	20.7652	4	76	54
7	7	92.3219	20.7652	9457	122	91
8	8	92.3303	20.7652	285	120	89
9	9	92.3135	20.7742	799	76	54
10	10	92.3219	20.7742	1611	13	5
11	11	92.3303	20.7742	3390	172	122
12	12	92.3051	20.7832	737	174	123
13	13	92.3135	20.7832	1796	174	123
14	14	92.3219	20.7832	344	121	90
15	15	92.3303	20.7832	730	121	90
16	16	92.3051	20.7921	8	88	63
17	17	92.3135	20.7921	6398	79	55
18	18	92.3219	20.7921	176	101	74

19	19	92.3303	20.7921	1008	101	74
20	20	92.2967	20.8011	28	114	84
21	21	92.3051	20.8011	2	114	84
22	23	92.3219	20.8011	184	101	74
23	24	92.3303	20.8011	570	185	124
24	25	92.3387	20.8011	6830	186	125
25	26	92.2967	20.8101	4581	115	85
26	27	92.3051	20.8101	702	114	84
27	30	92.3303	20.8101	286	127	95
28	31	92.3387	20.8101	2030	192	126
29	32	92.2883	20.8191	149	99	72
30	33	92.2967	20.8191	38	57	39
31	37	92.3303	20.8191	1113	124	92
32	38	92.2883	20.8281	1502	99	72
33	39	92.2967	20.8281	2493	116	86
34	42	92.3219	20.8281	2931	127	95
35	43	92.3303	20.8281	27	124	92
36	44	92.2799	20.8371	1428	92	66
37	45	92.2883	20.8371	1500	1	1
38	49	92.3219	20.8371	187	124	92
39	50	92.3303	20.8371	3418	126	94
40	51	92.2799	20.8461	319	1	1
41	52	92.2883	20.8461	99	1	1
42	56	92.3219	20.8461	3923	83	59
43	57	92.2715	20.8551	6748	95	68
44	58	92.2799	20.8551	4002	98	71
45	63	92.3219	20.8551	730	83	59
46	64	92.2631	20.8641	60	108	79
47	65	92.2715	20.8641	10529	108	79
48	70	92.3135	20.8641	11	125	93
49	71	92.3219	20.8641	7614	135	99
50	72	92.2631	20.8731	88	73	51
51	73	92.2715	20.8731	2119	94	67
52	77	92.3051	20.8731	9	96	69
53	78	92.3135	20.8731	110	70	50
54	79	92.2547	20.8821	2	73	51
55	80	92.2631	20.8821	279	133	97
56	81	92.2715	20.8821	480	133	97
57	85	92.3051	20.8821	71	96	69
58	86	92.3135	20.8821	1071	134	98
59	87	92.2463	20.8911	300	73	51
60	88	92.2547	20.8911	9	151	110
61	89	92.2631	20.8911	1818	113	83
62	92	92.2883	20.8911	1647	133	97
63	93	92.2967	20.8911	28	133	97
64	94	92.3051	20.8911	11	133	97
65	96	92.2463	20.9001	323	75	53
66	97	92.2547	20.9001	494	111	81
67	100	92.2799	20.9001	489	111	81

68	101	92.2883	20.9001	506	113	83
69	102	92.2967	20.9001	9	133	97
70	103	92.2295	20.9091	75	73	51
71	104	92.2379	20.9091	1707	151	110
72	105	92.2463	20.9091	951	75	53
73	108	92.2715	20.9091	481	111	81
74	109	92.2799	20.9091	2603	112	82
75	110	92.2883	20.9091	2042	103	76
76	112	92.2379	20.918	195	81	57
77	113	92.2463	20.918	3488	109	80
78	116	92.2715	20.918	31	111	81
79	117	92.2799	20.918	2489	86	61
80	118	92.2211	20.927	423	82	58
81	119	92.2295	20.927	21	82	58
82	120	92.2379	20.927	1540	81	57
83	123	92.2631	20.927	1027	104	77
84	124	92.2715	20.927	1438	104	77
85	125	92.2211	20.936	278	46	31
86	126	92.2295	20.936	969	46	31
87	127	92.2379	20.936	613	46	31
88	129	92.2547	20.936	231	105	78
89	130	92.2631	20.936	87	100	73
90	131	92.2715	20.936	1	46	31
91	132	92.2127	20.945	1103	131	96
92	133	92.2211	20.945	17	46	31
93	134	92.2295	20.945	840	46	31
94	137	92.2547	20.945	683	105	78
95	138	92.2631	20.945	284	100	73
96	139	92.2127	20.954	953	82	58
97	140	92.2211	20.954	1	102	75
98	141	92.2295	20.954	1190	102	75
99	144	92.2547	20.954	324	105	78
100	145	92.2631	20.954	511	100	73
101	146	92.2043	20.963	3085	131	96
102	147	92.2127	20.963	331	84	60
103	148	92.2211	20.963	78	84	60
104	152	92.2547	20.963	557	100	73
105	153	92.2631	20.963	107	100	73
106	154	92.2043	20.972	13	84	60
107	155	92.2127	20.972	71	84	60
108	156	92.2211	20.972	62	84	60
109	161	92.2631	20.972	1083	100	73
110	162	92.2043	20.981	67	84	60
111	163	92.2127	20.981	93	84	60
112	168	92.2547	20.981	4	100	73
113	169	92.2631	20.981	2	91	65
114	170	92.1959	20.99	523	67	48
115	171	92.2043	20.99	440	84	60
116	172	92.2127	20.99	6	84	60

117	177	92.2547	20.99	61	100	73
118	178	92.2631	20.99	52	91	65
119	179	92.1959	20.999	568	82	58
120	180	92.2043	20.999	320	84	60
121	181	92.2127	20.999	5	84	60
122	185	92.2463	20.999	1220	102	75
123	186	92.2547	20.999	189	105	78
124	187	92.2631	20.999	38	91	65
125	188	92.1959	21.008	225	74	52
126	189	92.2043	21.008	390	87	62
127	193	92.2379	21.008	55	84	60
128	194	92.2463	21.008	9	100	73
129	195	92.2547	21.008	3	91	65
130	196	92.1875	21.017	2	74	52
131	197	92.1959	21.017	260	74	52
132	198	92.2043	21.017	3	87	62
133	202	92.2379	21.017	80	84	60
134	203	92.2463	21.017	5	100	73
135	205	92.1791	21.026	1	67	48
136	206	92.1875	21.026	6	74	52
137	207	92.1959	21.026	312	87	62
138	208	92.2043	21.026	36	87	62
139	212	92.2379	21.026	124	102	75
140	213	92.2463	21.026	2	102	75
141	214	92.1791	21.035	187	74	52
142	215	92.1875	21.035	92	67	48
143	216	92.1959	21.035	156	87	62
144	221	92.2379	21.035	95	102	75
145	222	92.2463	21.035	1751	91	65
146	223	92.1707	21.044	251	67	48
147	224	92.1791	21.044	43	67	48
148	225	92.1875	21.044	237	67	48
149	231	92.2379	21.044	63	102	75
150	232	92.2463	21.044	162	102	75
151	233	92.2547	21.044	1612	84	60
152	234	92.1623	21.0529	503	67	48
153	235	92.1707	21.0529	76	67	48
154	236	92.1791	21.0529	1	67	48
155	243	92.2379	21.0529	6	84	60
156	244	92.2463	21.0529	120	84	60
157	245	92.2547	21.0529	120	87	62
158	246	92.1539	21.0619	137	97	70
159	247	92.1623	21.0619	639	97	70
160	248	92.1707	21.0619	8	97	70
161	257	92.2463	21.0619	10	87	62
162	258	92.2547	21.0619	108	87	62
163	259	92.2631	21.0619	94	87	62
164	260	92.1456	21.0709	108	97	70
165	261	92.1539	21.0709	191	97	70

166	262	92.1623	21.0709	3	97	70
167	271	92.2379	21.0709	6	87	62
168	272	92.2463	21.0709	19	87	62
169	273	92.2547	21.0709	113	87	62
170	274	92.1372	21.0799	1	117	87
171	275	92.1456	21.0799	626	117	87
172	276	92.1539	21.0799	2	97	70
173	284	92.2211	21.0799	116	87	62
174	285	92.2295	21.0799	5	87	62
175	286	92.2379	21.0799	126	87	62
176	287	92.2463	21.0799	32	87	62
177	289	92.1288	21.0889	20	154	112
178	290	92.1372	21.0889	6	154	112
179	291	92.1456	21.0889	162	117	87
180	292	92.1539	21.0889	385	97	70
181	300	92.2211	21.0889	27	87	62
182	301	92.2295	21.0889	15	87	62
183	302	92.2379	21.0889	13	87	62
184	303	92.1204	21.0979	3	146	105
185	304	92.1288	21.0979	75	45	30
186	305	92.1372	21.0979	6	45	30
187	306	92.1456	21.0979	10457	7	3
188	313	92.2043	21.0979	748	24	16
189	314	92.2127	21.0979	38	24	16
190	315	92.2211	21.0979	242	87	62
191	316	92.2295	21.0979	69	87	62
192	317	92.112	21.1069	329	146	105
193	318	92.1204	21.1069	12	45	30
194	319	92.1288	21.1069	302	45	30
195	320	92.1372	21.1069	188	45	30
196	327	92.1959	21.1069	979	24	16
197	328	92.2043	21.1069	57	119	88
198	329	92.2127	21.1069	118	24	16
199	330	92.2211	21.1069	78	87	62
200	331	92.1036	21.1159	1425	54	37
201	332	92.112	21.1159	374	45	30
202	333	92.1204	21.1159	447	45	30
203	334	92.1288	21.1159	560	11	4
204	341	92.1875	21.1159	1217	5	2
205	342	92.1959	21.1159	56	80	56
206	343	92.2043	21.1159	1179	89	64
207	344	92.2127	21.1159	4719	24	16
208	345	92.2211	21.1159	67	87	62
209	346	92.0952	21.1249	179	147	106
210	347	92.1036	21.1249	3	61	43
211	348	92.112	21.1249	2208	61	43
212	349	92.1204	21.1249	340	11	4
213	356	92.1791	21.1249	82	5	2
214	357	92.1875	21.1249	43	89	64

215	358	92.1959	21.1249	455	161	117
216	359	92.0868	21.1339	278	61	43
217	360	92.0952	21.1339	31	61	43
218	361	92.1036	21.1339	333	66	47
219	370	92.1791	21.1339	20	29	19
220	371	92.1875	21.1339	1790	30	20
221	372	92.1959	21.1339	469	30	20
222	373	92.0868	21.1429	4	69	49
223	374	92.0952	21.1429	1212	61	43
224	375	92.1036	21.1429	145	30	20
225	382	92.1623	21.1429	6665	36	24
226	383	92.1707	21.1429	1201	5	2
227	384	92.1791	21.1429	1	29	19
228	385	92.1875	21.1429	110	30	20
229	386	92.1959	21.1429	77	43	28
230	387	92.07	21.1519	25	156	113
231	388	92.0784	21.1519	876	69	49
232	389	92.0868	21.1519	16	69	49
233	390	92.0952	21.1519	153	40	27
234	398	92.1623	21.1519	484	38	26
235	399	92.1707	21.1519	1559	36	24
236	400	92.1791	21.1519	2	47	32
237	401	92.1875	21.1519	2129	43	28
238	402	92.1959	21.1519	3	43	28
239	403	92.07	21.1609	186	149	108
240	404	92.0784	21.1609	1	40	27
241	405	92.0868	21.1609	3188	37	25
242	406	92.0952	21.1609	1617	33	22
243	413	92.1539	21.1609	114	44	29
244	414	92.1623	21.1609	617	38	26
245	415	92.1707	21.1609	3501	59	41
246	416	92.1875	21.1609	234	26	18
247	417	92.1959	21.1609	1249	43	28
248	418	92.0616	21.1699	1435	153	111
249	419	92.07	21.1699	1070	37	25
250	420	92.0784	21.1699	301	34	23
251	421	92.0868	21.1699	2657	34	23
252	429	92.1539	21.1699	12	44	29
253	430	92.1623	21.1699	63	56	38
254	431	92.1707	21.1699	388	56	38
255	432	92.0532	21.1788	16	62	44
256	433	92.0616	21.1788	1186	34	23
257	434	92.07	21.1788	315	34	23
258	435	92.0784	21.1788	26	34	23
259	444	92.1539	21.1788	555	15	7
260	445	92.1623	21.1788	1	49	33
261	446	92.1707	21.1788	207	49	33
262	447	92.0532	21.1878	94	62	44
263	448	92.0616	21.1878	10	18	10

264	449	92.07	21.1878	544	18	10
265	459	92.1539	21.1878	52	22	14
266	460	92.1623	21.1878	41	49	33
267	461	92.1707	21.1878	491	49	33
268	462	92.0532	21.1968	27	18	10
269	463	92.0616	21.1968	474	19	11
270	464	92.07	21.1968	1751	19	11
271	474	92.1539	21.1968	85	22	14
272	475	92.1623	21.1968	1289	49	33
273	476	92.1707	21.1968	1192	14	6
274	477	92.0532	21.2058	2023	18	10
275	478	92.0616	21.2058	653	18	10
276	479	92.07	21.2058	193	19	11
277	488	92.1456	21.2058	40	64	46
278	489	92.1539	21.2058	718	22	14
279	490	92.1623	21.2058	146	49	33
280	491	92.0532	21.2148	2152	23	15
281	492	92.0616	21.2148	332	62	44
282	493	92.07	21.2148	2104	144	103
283	502	92.1456	21.2148	838	21	13
284	503	92.1539	21.2148	42	22	14
285	504	92.1623	21.2148	305	22	14
286	505	92.0532	21.2238	7	148	107
287	506	92.0616	21.2238	59	16	8
288	507	92.07	21.2238	1314	62	44
289	516	92.1456	21.2238	194	22	14
290	517	92.1539	21.2238	880	22	14
291	518	92.1623	21.2238	435	14	6
292	519	92.0532	21.2328	60	148	107
293	520	92.0616	21.2328	760	136	100
294	521	92.07	21.2328	1023	16	8
295	531	92.1539	21.2328	336	49	33
296	532	92.1623	21.2328	358	25	17
297	533	92.1707	21.2328	1262	25	17
298	534	92.0532	21.2418	2429	159	116
299	535	92.0616	21.2418	2215	145	104
300	536	92.07	21.2418	1254	23	15
301	547	92.1623	21.2418	198	25	17
302	548	92.1707	21.2418	440	32	21
303	549	92.1791	21.2418	62	32	21
304	550	92.0532	21.2508	151	159	116
305	551	92.0616	21.2508	22	158	115
306	552	92.07	21.2508	461	145	104
307	563	92.1623	21.2508	627	25	17
308	564	92.1707	21.2508	14	32	21
309	565	92.1791	21.2508	1677	14	6
310	566	92.0532	21.2598	748	159	116
311	567	92.0616	21.2598	11	158	115
312	568	92.07	21.2598	445	145	104

313	579	92.1623	21.2598	17	32	21
314	580	92.1707	21.2598	334	53	36
315	581	92.1791	21.2598	88	53	36
316	582	92.1875	21.2598	6	53	36
317	583	92.0532	21.2688	1	159	116
318	584	92.0616	21.2688	272	158	115
319	585	92.07	21.2688	764	50	34
320	586	92.0784	21.2688	23	145	104
321	596	92.1623	21.2688	231	53	36
322	597	92.1707	21.2688	6	53	36
323	598	92.1791	21.2688	51	53	36
324	599	92.1875	21.2688	1157	53	36
325	600	92.0532	21.2778	28	159	116
326	601	92.0616	21.2778	396	158	115
327	602	92.07	21.2778	259	157	114
328	603	92.0784	21.2778	43	150	109
329	604	92.0868	21.2778	3299	58	40
330	612	92.1539	21.2778	160	14	6
331	613	92.1623	21.2778	197	14	6
332	614	92.1707	21.2778	1421	17	9
333	615	92.0532	21.2868	64	159	116
334	616	92.0616	21.2868	1559	158	115
335	617	92.07	21.2868	341	50	34
336	618	92.0784	21.2868	550	50	34
337	619	92.0868	21.2868	508	50	34
338	620	92.0952	21.2868	27	58	40
339	621	92.1036	21.2868	1035	136	100
340	626	92.1456	21.2868	393	22	14
341	627	92.1539	21.2868	666	17	9
342	628	92.1623	21.2868	46	17	9
343	629	92.1707	21.2868	24	17	9
344	630	92.0784	21.2958	94	50	34
345	631	92.0868	21.2958	154	50	34
346	632	92.0952	21.2958	333	50	34
347	633	92.1036	21.2958	23	144	103
348	634	92.112	21.2958	1	144	103
349	638	92.1456	21.2958	35	20	12
350	639	92.1539	21.2958	235	17	9
351	640	92.1623	21.2958	1	17	9
352	641	92.1707	21.2958	170	17	9
353	642	92.0868	21.3047	83	50	34
354	643	92.0952	21.3047	1	50	34
355	644	92.1036	21.3047	4432	139	102
356	646	92.1204	21.3047	365	60	42
357	647	92.1288	21.3047	172	60	42
358	648	92.1372	21.3047	198	21	13
359	649	92.1456	21.3047	16	20	12
360	650	92.1539	21.3047	79	20	12
361	651	92.1623	21.3047	34	17	9

362	652	92.1036	21.3137	420	137	101
363	653	92.112	21.3137	1129	144	103
364	654	92.1204	21.3137	459	60	42
365	655	92.1288	21.3137	2125	60	42
366	656	92.1372	21.3137	205	21	13
367	657	92.1456	21.3137	1135	51	35
368	658	92.1539	21.3137	1	20	12
369	659	92.112	21.3227	452	137	101
370	660	92.1204	21.3227	39	60	42
371	661	92.1288	21.3227	129	21	13
372	662	92.1372	21.3227	9	21	13
373	663	92.1456	21.3227	1149	51	35
374	664	92.1539	21.3227	272	20	12
375	665	92.1204	21.3317	1210	63	45
376	666	92.1288	21.3317	89	63	45
377	667	92.1372	21.3317	794	63	45
378	668	92.1456	21.3317	753	21	13
379	669	92.1288	21.3407	838	63	45
380	670	92.1372	21.3407	87	63	45
381	671	92.1456	21.3407	130	63	45

Summary by dominant shelter:

126×6 DataFrame

Row	dom_j	n_cells	dominated_pop	lon	lat	is_new
	Int64	Int64	Int64	Float64	Float64	Bool
1	108	2	10589	92.2157	20.9421	false
2	7	1	10457	92.1615	21.1443	false
3	122	1	9457	92.3237	20.7641	false
4	36	2	8224	92.1571	21.2412	false
5	135	1	7614	92.2986	20.8739	false
6	186	1	6830	92.3387	20.8011	true
7	95	1	6748	92.2528	20.8889	false
8	24	5	6602	92.2113	21.0911	false
9	79	1	6398	92.3148	20.8005	false
10	163	1	5689	92.3387	20.7472	true
11	83	2	4653	92.3041	20.8646	false
12	115	1	4581	92.3002	20.8285	false
13	34	5	4485	92.1141	21.261	false
14	139	1	4432	92.0979	21.3044	false
15	37	2	4258	92.1312	21.2496	false
16	131	2	4188	92.1777	21.032	false
17	98	1	4002	92.2796	20.856	false
18	61	5	3732	92.1571	21.1705	false
19	14	5	3661	92.1459	21.2815	false
20	59	1	3501	92.1488	21.2513	false
21	109	1	3488	92.2373	21.0149	false
22	43	4	3458	92.1703	21.2362	false
23	159	6	3421	92.0544	21.2864	false
24	126	1	3418	92.296	20.8603	false

25	23	2	3406	92.0989	21.2724	false
26	172	1	3390	92.3303	20.7742	true
27	84	16	3359	92.2256	21.0605	false
28	58	2	3326	92.092	21.2873	false
29	18	5	3257	92.1113	21.273	false
30	144	4	3257	92.1203	21.2824	false
31	166	1	3244	92.3387	20.7562	true
32	127	2	3217	92.2956	20.8602	false
33	60	5	3160	92.1307	21.2892	false
34	63	6	3148	92.1328	21.3326	false
35	145	4	3144	92.0956	21.2773	false
36	102	8	2857	92.2333	21.0515	false
37	50	9	2828	92.0834	21.2987	false
38	46	6	2718	92.2342	21.0282	false
39	100	10	2708	92.2406	21.0321	false
40	22	8	2669	92.1384	21.2845	false
41	112	1	2603	92.2414	20.9895	false
42	17	8	2597	92.1559	21.2895	false
43	174	2	2533	92.3135	20.7832	true
44	30	4	2514	92.1619	21.223	false
45	49	7	2511	92.1436	21.2674	false
46	5	3	2500	92.1698	21.1448	false
47	116	1	2493	92.2999	20.8289	false
48	86	1	2489	92.2517	21.0127	false
49	104	2	2465	92.2425	21.019	false
50	133	6	2454	92.249	20.9723	false
51	25	4	2445	92.152	21.268	false
52	19	3	2418	92.1222	21.275	false
53	113	2	2324	92.2421	20.9873	false
54	51	2	2284	92.1434	21.3063	false
55	158	5	2260	92.0552	21.2858	false
56	87	23	2157	92.217	21.0775	false
57	21	6	2132	92.1375	21.2846	false
58	94	1	2119	92.2504	20.9658	false
59	103	1	2042	92.2548	20.9981	false
60	192	1	2030	92.3387	20.8101	true
61	82	4	1965	92.188	21.0158	false
62	1	3	1918	92.2907	20.8569	false
63	53	7	1873	92.1588	21.2746	false
64	91	5	1846	92.2458	21.0417	false
65	136	2	1795	92.1052	21.2817	false
66	62	4	1756	92.1036	21.2638	false
67	81	2	1735	92.1916	21.0031	false
68	67	9	1727	92.1625	21.0553	false
69	151	2	1716	92.2019	20.9742	false
70	99	2	1651	92.2926	20.8309	false
71	33	1	1617	92.1257	21.2551	false
72	13	1	1611	92.3261	20.7729	false
73	111	4	1495	92.2409	20.9897	false

74	97	8	1473	92.1541	21.0652	false
75	153	1	1435	92.049	21.2102	false
76	92	1	1428	92.2812	20.8436	false
77	105	4	1427	92.2402	21.0335	false
78	54	1	1425	92.1122	21.11	false
79	45	7	1404	92.1215	21.1029	false
80	101	3	1368	92.3146	20.8084	false
81	124	3	1327	92.2973	20.8589	false
82	75	2	1274	92.1993	20.985	false
83	89	2	1222	92.1845	21.1326	false
84	162	1	1211	92.3303	20.7472	true
85	38	2	1101	92.1376	21.2459	false
86	16	2	1082	92.1108	21.2733	false
87	121	2	1074	92.3348	20.7817	false
88	134	1	1071	92.2555	20.951	false
89	11	2	900	92.1579	21.1703	false
90	69	3	896	92.1201	21.169	false
91	137	2	872	92.0974	21.3112	false
92	76	2	803	92.3196	20.7687	false
93	117	3	789	92.1443	21.0741	false
94	114	3	732	92.3061	20.8184	false
95	74	5	680	92.1772	21.0378	false
96	185	1	570	92.3303	20.8011	true
97	15	1	555	92.1342	21.2745	false
98	32	4	533	92.1589	21.2625	false
99	73	4	465	92.2079	20.9553	false
100	161	1	455	92.1771	21.1467	false
101	56	2	451	92.1365	21.2585	false
102	20	5	403	92.1437	21.2954	false
103	66	1	333	92.137	21.1555	false
104	146	2	332	92.1197	21.1006	false
105	120	1	285	92.3311	20.7668	false
106	157	1	259	92.0707	21.2841	false
107	165	2	239	92.3303	20.7562	true
108	26	1	234	92.1657	21.2568	false
109	149	1	186	92.0496	21.2362	false
110	147	1	179	92.0952	21.125	false
111	40	2	154	92.1408	21.2373	false
112	44	2	126	92.1262	21.2557	false
113	70	1	110	92.305	20.874	false
114	96	2	80	92.2957	20.8749	false
115	148	2	67	92.0507	21.273	false
116	119	1	57	92.2057	21.1083	false
117	80	1	56	92.1957	21.1158	false
118	150	1	43	92.0811	21.288	false
119	64	1	40	92.1246	21.2684	false
120	57	1	38	92.2962	20.8249	false
121	154	2	26	92.1298	21.0914	false
122	156	1	25	92.0571	21.1705	false
123	29	2	21	92.1618	21.2223	false

```
 124 │   125        1             11  92.2967  20.8725   false
 125 │    88        1              8  92.3078  20.8024   false
 126 │    47        1              2  92.1592  21.2486   false
```

```
# new_coords_df: the indices + (lon, lat) of all new shelters opened
in the K* solution.

# masks
new_mask = [j in JN for j in 1:nJ]          # true if candidate-new
shelter
existing_mask = .!new_mask                  # true if original existing
shelter

# open vectors
open_base = round.(Int, sol_base.y) .== 1
open_star = round.(Int, sol_star.y) .== 1

# New shelters opened at K*
open_new_star = open_star .& new_mask
new_inds = findall(open_new_star)

new_coords_df = DataFrame(
    shelter_index = new_inds,
    lon = lonJ[new_inds],
    lat = latJ[new_inds]
)

println("New shelters opened in K* = $K_star")
show(new_coords_df, allrows=true, allcols=true)
```

```
New shelters opened in K* = 10
10×3 DataFrame
 Row │ shelter_index  lon      lat
     │ Int64          Float64  Float64
─────┼──────────────────────────────────
   1 │           162  92.3303  20.7472
   2 │           163  92.3387  20.7472
   3 │           165  92.3303  20.7562
   4 │           166  92.3387  20.7562
   5 │           168  92.3219  20.7652
   6 │           172  92.3303  20.7742
   7 │           174  92.3135  20.7832
   8 │           185  92.3303  20.8011
   9 │           186  92.3387  20.8011
  10 │           192  92.3387  20.8101
```

```
# closed_existing_df: which baseline-existing shelters (open at K=0)
are no longer open at K*.

# Existing shelters open at K=0 but NOT used/open in K* solution
open_existing_base = open_base .& existing_mask
```

```
open_existing_star = open_star .& existing_mask

existing_base_inds = findall(open_existing_base)
existing_star_inds = findall(open_existing_star)

# Existing shelters that were open in baseline but are CLOSED in K*
closed_existing_inds = setdiff(existing_base_inds, existing_star_inds)

closed_existing_df = DataFrame(
    shelter_index = closed_existing_inds,
    lon = lonJ[closed_existing_inds],
    lat = latJ[closed_existing_inds]
)

println("Existing shelters open at K=0 but CLOSED at K*")
show(closed_existing_df, allrows=true, allcols=true)
```

```
Existing shelters open at K=0 but CLOSED at K*
0×3 DataFrame
 Row │ shelter_index  lon      lat
     │ Int64          Float64  Float64
─────┴──────────────────────────────────
```

```
# unused_existing_df: baseline-existing shelters that are effectively
not used at K* bc they get ~0 assigned population, even if open.

# Check "not used" via assignments, not just open
# (open but zero assigned population)
pop_assigned_star = [
    sum(d[i] * sol_star.x[i,j] for i in 1:nI) for j in 1:nJ
]

used_star_mask = pop_assigned_star .> 1e-6

# existing shelters that were open at K=0 but receive ~0 pop at K*
unused_existing_inds = [
    j for j in existing_base_inds if !used_star_mask[j]
]

unused_existing_df = DataFrame(
    shelter_index = unused_existing_inds,
    lon = lonJ[unused_existing_inds],
    lat = latJ[unused_existing_inds],
    pop_assigned = pop_assigned_star[unused_existing_inds]
)

println("Existing shelters open at K=0 but UNUSED at K* (pop≈0)")
show(unused_existing_df, allrows=true, allcols=true)
```

```
Existing shelters open at K=0 but UNUSED at K* (pop≈0)
0×4 DataFrame
```

```
 Row │ shelter_index  lon      lat      pop_assigned
     │ Int64          Float64  Float64  Float64
─────┼──────────────────────────────────────────────
```