

```

function [assignments_primary, assignments_dual, cost_primary, cost_dual]
= assign_users_munkres_new( ...
    W, prev_assign, prev_assign_dual, G, ...
    alpha, beta, threshold, low_quality_threshold, ...
    first_assignment, use_dual)
% assign_users_munkres (fully modular, capacity enforced for both primary
and dual)
%
% Inputs:
%   W                - (U x M) metric matrix (SNR [dB] or rate)
%   prev_assign       - (U x 1) previous primary assignment
%   prev_assign_dual   - (U x 1) previous dual assignment
%   G                 - max users per satellite
%   alpha, beta       - switching penalties
%   threshold, low_quality_threshold - for dual repair
%   first_assignment   - true at t=1 (no penalty)
%   use_dual          - false: do primary only, true: do dual only
%
% Outputs:
%   assignments_primary - (U x 1) new primary (if primary step)
%   assignments_dual    - (U x 1) new dual (if dual step)
%   cost_primary        - primary assignment cost
%   cost_dual           - dual handover cost

[U, M] = size(W);
assignments_primary = prev_assign; % Default: unchanged
assignments_dual    = prev_assign_dual; % Default: unchanged
cost_primary = 0;
cost_dual = 0;

if ~use_dual
    % =====
    % === Primary only! ===
    % =====
    W_penalized = W;
    if ~first_assignment
        for u = 1:U
            if prev_assign(u) > 0 && prev_assign(u) <= M
                W_penalized(u, prev_assign(u)) = W_penalized(u,
prev_assign(u)) - alpha;
            end
        end
    end
end

% --- Enforce satellite capacity (count both primary and dual
assignments) ---
all_assigned = [prev_assign(:); prev_assign_dual(:)];
sat_load = histcounts(all_assigned(all_assigned > 0), 1:M+1);
available_slots = G - sat_load;
for m = 1:M
    if available_slots(m) <= 0
        W_penalized(:, m) = 0; % Block assignments to full satellites
    end
end

valid_users = any(W_penalized, 2);
valid_sats = any(W_penalized, 1);
if ~any(valid_users) || ~any(valid_sats)

```

```

        assignments_primary = zeros(U, 1);
        return;
    end

    user_idx_map = find(valid_users);
    sat_idx_map = find(valid_sats);
    W_small = W_penalized(valid_users, valid_sats);

    % Expand satellite capacity for Munkres (based on actual available
slots)
    W_expanded = [];
    sat_idx_expanded = [];
    for i = 1:length(sat_idx_map)
        m = sat_idx_map(i);
        n_slots = available_slots(m);
        if n_slots > 0
            W_expanded = [W_expanded, repmat(W_small(:,i), 1, n_slots)];
            sat_idx_expanded = [sat_idx_expanded, repmat(m, 1, n_slots)];
        end
    end

    % Primary assignment using Hungarian algorithm
    [raw_assignment, raw_cost] = munkres(-W_expanded);

    assignments_reduced = zeros(length(user_idx_map), 1);
    valid_idx = raw_assignment > 0;
    assignments_reduced(valid_idx) =
sat_idx_expanded(raw_assignment(valid_idx));

    assignments_primary = zeros(U, 1);
    assignments_primary(user_idx_map) = assignments_reduced;
    cost_primary = -raw_cost;

    % Dual remains as input (unchanged)
    assignments_dual = prev_assign_dual;
    cost_dual = 0;
    return;
end

% =====
% ==== Dual only! =====
% =====
% Use prev_assign as the current primary assignment

% 1. Identify users needing repair (poor or no assignment)
assigned_metrics = zeros(U, 1);
assigned = prev_assign > 0;
assigned_metrics(assigned) = W(sub2ind(size(W), find(assigned),
prev_assign(assigned)));
repair_users = find(assigned_metrics < low_quality_threshold);

assignments_primary = prev_assign;          % (carry over)
assignments_dual = prev_assign_dual;        % <-- HOLD previous dual
assignments!
cost_primary = 0;                          % not updated in dual mode

if isempty(repair_users)
    return;

```

```

end

% 2. Satellites with available slots (count both primary and dual)
all_assigned = [prev_assign(:); prev_assign_dual(:)];
sat_load = histcounts(all_assigned(all_assigned > 0), 1:M+1);
available_slots = G - sat_load;
available_sats = find(available_slots > 0);
if isempty(available_sats)
    return;
end

% 3. Build dual repair matrix
W_repair = W(repair_users, available_sats);
W_repair(W_repair < threshold) = 0;
valid_users_dual = any(W_repair, 2);
repair_users = repair_users(valid_users_dual);
W_repair = W_repair(valid_users_dual, :);
valid_sats_dual = any(W_repair, 1);
available_sats = available_sats(valid_sats_dual);
W_repair = W_repair(:, valid_sats_dual);

if isempty(repair_users) || isempty(available_sats)
    return;
end

% 4. Dual penalties
if ~first_assignment
    for i = 1:length(repair_users)
        u = repair_users(i);
        prev_dual = prev_assign_dual(u);
        if prev_dual > 0
            local_idx = find(available_sats == prev_dual);
            if ~isempty(local_idx)
                W_repair(i, local_idx) = W_repair(i, local_idx) - beta;
            end
        end
    end
end

% 5. Satellite capacity expansion for dual
sat_repeat = arrayfun(@(s) repmat(s, 1, available_slots(s)), ...
    available_sats, 'UniformOutput', false);
sat_repeat = [sat_repeat{:}];
W_expanded_repair = [];
for i = 1:length(available_sats)
    col = W_repair(:, i);
    reps = available_slots(available_sats(i));
    W_expanded_repair = [W_expanded_repair, repmat(col, 1, reps)];
end

% 6. Munkres for dual
[repair_assignment, cost_dual_raw] = munkres(-W_expanded_repair);
valid_repair = repair_assignment > 0;
repair_slots = repair_assignment(valid_repair);
repair_users_final = repair_users(valid_repair);
repair_sats = sat_repeat(repair_slots);

```

```
assignments_dual(repair_users_final) = repair_sats;  
cost_dual = -cost_dual_raw;  
  
end
```