

```

function [assignments_primary, assignments_dual, cost_primary, cost_dual]
= assign_users_munkres_algo_1( ...
    W, prev_assign, prev_assign_dual, G, ...
    alpha, beta, threshold, low_quality_threshold, ...
    first_assignment, use_dual)
% assign_users_munkres
% Unified assignment function with dual-handover support and cost
breakdown.
%
% 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             - penalty for switching primary
%   beta              - penalty for switching dual
%   threshold         - minimum link value for repair
%   low_quality_threshold - quality below which dual reassignment is
triggered
%   first_assignment  - true at t = 1 (disables penalties)
%   use_dual          - enable dual-handover step
%
% Outputs:
%   assignments_primary - (U x 1) initial primary assignment before
repair
%   assignments_dual    - (U x 1) repair assignment only (0 if
unchanged)
%   cost_primary        - primary assignment cost
%   cost_dual           - dual handover cost

[U, M] = size(W);
assignments_primary = zeros(U, 1);
assignments_dual = zeros(U, 1);
cost_primary = 0;
cost_dual = 0;

%%Step 1: Apply primary handover penalties
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

%%Step 2: Filter valid users and satellites
valid_users = any(W_penalized, 2);
valid_sats = any(W_penalized, 1);
if ~any(valid_users) || ~any(valid_sats)
    return;
end

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

```

```

%%Step 3: Expand satellite capacity for Munkres
W_expanded = repmat(W_small, 1, G); % (U' x M'*G)

%%Step 4: Primary assignment using Hungarian algorithm
[raw_assignment, raw_cost] = munkres(-W_expanded);
slot_idx = ceil(raw_assignment / G);
assignments_reduced = zeros(length(user_idx_map), 1);
valid_idx = raw_assignment > 0;
assignments_reduced(valid_idx) = sat_idx_map(slot_idx(valid_idx));
assignments_primary(user_idx_map) = assignments_reduced;

% Record cost from primary assignment
cost_primary = -raw_cost;
cost_total = cost_primary;

%%Step 5: Dual-handover repair (optional)
if ~use_dual
    return;
end

% Step 5.1: Identify users with poor or no assignment
assigned_metrics = zeros(U, 1);
assigned = assignments_primary > 0;
assigned_metrics(assigned) = W(sub2ind(size(W), find(assigned),
assignments_primary(assigned)));
repair_users = find(assigned_metrics < low_quality_threshold);
if isempty(repair_users)
    return;
end

% Step 5.2: Find satellites with available slots
sat_load = histcounts(assignments_primary(assignments_primary > 0),
1:M+1);
available_slots = G - sat_load;
available_sats = find(available_slots > 0);
if isempty(available_sats)
    return;
end

% Step 5.3: Build repair matrix
W_repair = W(repair_users, available_sats);
W_repair(W_repair < threshold) = 0;

% Remove users who have no valid satellite options in W_repair
valid_users_dual = any(W_repair, 2);
repair_users = repair_users(valid_users_dual);
W_repair = W_repair(valid_users_dual, :);

% Remove satellites (columns) that have no value left
valid_sats_dual = any(W_repair, 1);
available_sats = available_sats(valid_sats_dual);
W_repair = W_repair(:, valid_sats_dual);

% Re-check size
if isempty(repair_users) || isempty(available_sats)
    return;
end

```

```

% Step 5.4: Apply dual-handover 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
end

```

```

% Step 5.5: Expand for remaining satellite capacity
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

```

```

% Step 5.6: Run Munkres again for repair users
[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);

```

```

% Update dual and final assignments
assignments_dual(repair_users_final) = repair_sats;

```

```

% Compute cost: consistent with penalized assignment
cost_dual = -cost_dual_raw;

```

```

end

```

```

% -----
%
% SUMMARY OF THIS FUNCTION (assign_users_munkres)
%
% This function performs user-to-satellite assignment for each time slot,
% supporting both primary assignment and a dual-handover (repair) step:
%
% 1. Primary Assignment:
%    - Applies penalties to discourage unnecessary switching of user-
%      satellite assignments.
%    - Builds an assignment metric (SNR or rate) for all user-satellite
%      pairs.
%    - Uses the Hungarian algorithm to assign users to satellites,
%      expanding satellites'

```

```

%      capacity as needed to allow multiple users per satellite (up to
G).
%
% 2. Dual-Handover/Repair Step:
%   - Identifies users with low-quality or failed primary assignments.
%   - Searches for additional satellites with available capacity for
these users.
%   - Uses a second Hungarian optimization (with dual-switch penalties)
to assign each
%       "repair user" to a new satellite if possible, avoiding overload.
%
% 3. Cost Outputs:
%   - Returns the total assignment "cost" (e.g., SNR, rate, or a
weighted sum) for both
%       the primary and dual assignments, including all penalties.
%
% The function's output enables dual-connectivity handover strategies in
LEO satellite
% networks, supporting robust and fair performance evaluation under
realistic constraints.
% -----
-

```