Optimization Competition

—Based on *Greedy Algorithm* and connectivity penalty

Jia-Xin Wang

China-UK Low Carbon College, Shanghai Jiao Tong University, Shanghai, China

May 24, 2025

Shanghai Jiao Tong University

Overview

1. Optimization objectives and constraints
2. Algorithm
3. Result

Optimization objectives and constraints

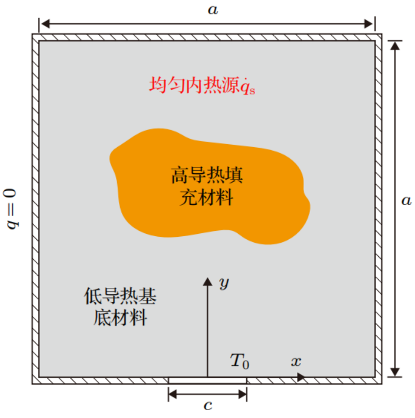


图 1　体点导热问题示意图

Fig. 1. The schematic diagram of the VP problem.

Find the minimum value of a function

$$T^* = \frac{k_0(T - T_0)}{L_x^2 q}$$

$$\overline{T^*} = \frac{1}{N}\sum_{i=1}^{N} T_i^*$$

Constraints

$$\sum_{i=1}^{N} \mathbb{I}(k_i > 250) \leq 0.15N$$

```
penalty_coeff*(num_components-1)
```

Optimization objectives and constraints

Mathematical Formulation of the Complete Optimization Problem

$$\min_{k} \left(\overline{T^*} + \lambda \cdot (\text{num\_components} - 1) \right)$$

$$\text{subject to: } \sum_{i=1}^{N} \mathbb{I}(k_i = k_1) \leq 0.15N$$

λ: Penalty coefficient (dynamically adjusted, see `penalty_coeff` in the code)
$k_i \in \{k_0, k_1\}$: Unit thermal conductivity (binary distribution, $k_0 = 1.0$, $k_1 = 500.0$)

Algorithm

Greedy Algorithm

Local Optimality: At each step, only the best current choice is considered, without backtracking or global consideration of future impacts.
Example: When making change, always use the largest denomination coin first.
No Aftereffect: Current choices do not affect the structure of subsequent subproblems (i.e., subproblems are independent).
Example: When selecting paths, the current path choice doesn't change weights of subsequent paths.
High Efficiency: Typically has low time complexity (e.g., $O(n \log n)$), making it suitable for large-scale problems.
No Global Optimality Guarantee: Since only local optima are considered, the final result may be an approximate solution (though optimal for certain specific problems).

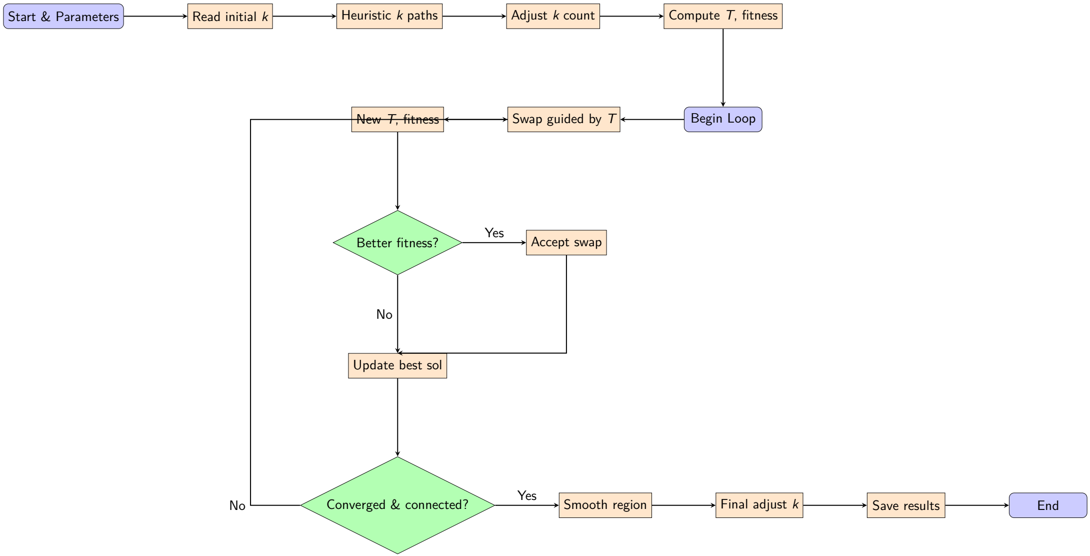
Algorithm

Optimization problem characteristics:

- **Discrete decision space** (Binary thermal conductivity distribution $k_i \in \{k_0, k_1\}$)
- **Decomposable local effects** (Changes in single unit's conductivity mainly affect neighboring region's temperature)
- **Submodularity property**: Diminishing marginal returns when adding high-conductivity material:

$$\Delta T^*(S \cup \{i\}) - \Delta T^*(S) \geq \Delta T^*(T \cup \{i\}) - \Delta T^*(T), \quad \forall S \subseteq T$$

Algorithm flow chart



Algorithm code snap

New function: Calculate fitness with penalty term

```
% New function: Calculate fitness with penalty term
function fitness = calculate_fitness_with_penalty(T_flipped, k_opt)
% Calculate average temperature
mean_temp = mean(T_flipped(:));
% Calculate connectivity penalty
k_binary = k_opt > 250;
cc = bwconncomp(k_binary, 8);
num_components = cc.NumObjects;
% Penalty coefficient (adjustable)
penalty_coeff = 0.001;
% Total fitness = average temp + connectivity penalty
fitness = mean_temp + penalty_coeff * (num_components - 1);
end
```

Algorithm code snap

Greedy Algorithm Settings

```
% Improved greedy algorithm (enhanced continuity constraints)
max_iter_greedy = 1000; % Maximum iterations
num_swaps = 10; % Number of exchange attempts per iteration
best_solution_greedy = current_solution;
best_fitness_greedy = current_fitness;
fitness_history_greedy = zeros(max_iter_greedy, 1);

for iter = 1:max_iter_greedy
    % Try multiple swaps and select the best
    idx1 = find(current_solution == 1); % High-conductivity cells
    idx0 = find(current_solution == 0); % Low-conductivity cells
    best_swap_fitness = current_fitness;
    best_swap_solution = current_solution;

    for s = 1:num_swaps
        % Randomly select cells to swap
        swap1 = idx1(randi(length(idx1)));
        swap0 = idx0(randi(length(idx0)));
        new_solution = current_solution;
        new_solution(swap1) = 0;
        new_solution(swap0) = 1;

        % Calculate new conductivity distribution
        k_opt = reshape(new_solution, [nx, ny]) * (k1 - k0) + k0;
```

Post-processing - Improved smoothing

```
% Smoothing process: Ensure continuous high-conductivity regions
k_binary = k_opt > 250; % Convert to binary matrix (1=high conductivity)

% Find largest connected region
cc = bwconncomp(k_binary, 8);
numPixels = cellfun(@numel, cc.PixelIdxList);
[~, idx] = max(numPixels);
k_smooth = false(size(k_binary));
k_smooth(cc.PixelIdxList{idx}) = true;

% Add isolated points adjacent to main region
for i = 2:nx-1
    for j = 2:ny-1
        if ~k_smooth(i,j) && k_binary(i,j)
            % Check if adjacent to main region
            neighbors = k_smooth(i-1:i+1, j-1:j+1);
            if sum(neighbors(:)) > 0
                k_smooth(i,j) = true;
            end
        end
    end
end
```

Result

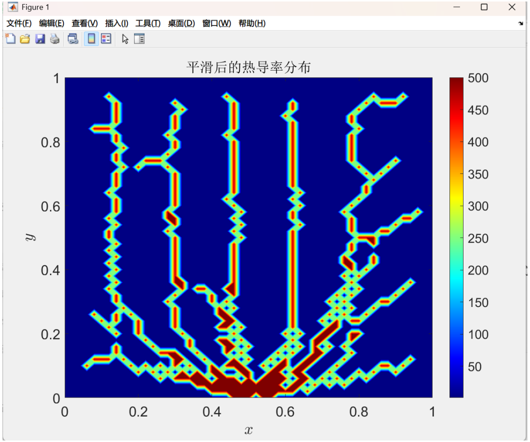


Figure: Thermal conductivity distribution after 3000 iterations

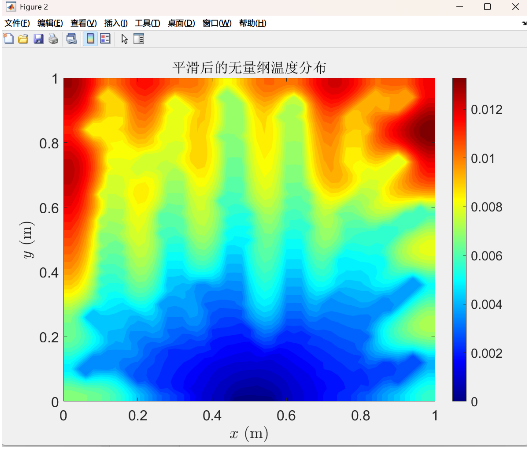


Figure: Temperature distribution after 3000 iterations (**0.006544**)

Python version

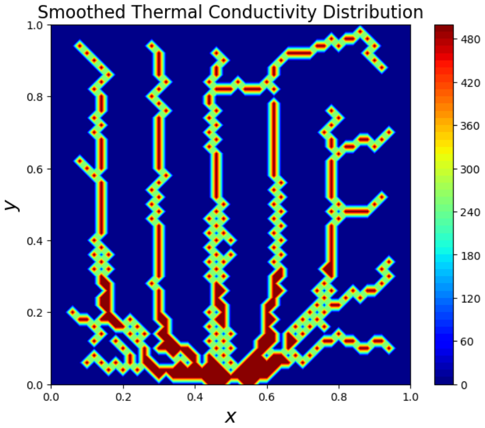


Figure: Thermal conductivity distribution after 10000 iterations (**0.006538**)

Accelerate:

- from **numba** import **njit**
- from **multiprocessing** import **Pool** (**Hints:** PSO and GA can be calculated in parallel, but other algorithms cannot.)

Github:

`https://github.com/wjx0209/Optimization_Competition.git`

The End