Hassan Imran 22I-0813 Section E

Artificial Intelligence

Assignment # 2 - Report

Graph Coloring Using Local Beam Search

Introduction

The problem involves coloring an undirected graph G=(V,E) such that:

- No two adjacent vertices share the same color (standard graph coloring constraint).
-
- The number of colors used is minimized (chromatic number constraint).

Additional constraints include:

- Higher-degree vertices are colored first.
- Some vertices have predefined colors that cannot be changed.
- Colors are distributed evenly across the graph (balanced color usage).
- Vertices two hops away must have different colors.

This report explains the steps involved in the program, the implementation of Local Beam Search, and the results obtained.

Steps Involved in the Program :

1. Reading the Graph and Predefined Colors

Input Files:

hypercube_dataset_orignal.txt: Contains the graph in the format Source Destination Heuristic.

predefined_colors.txt: Contains vertices with predefined colors in the format Vertex Color.

Functions:

read_graph(file_path): Reads the graph and constructs an adjacency list.

read_predefined_colors(file_path): Reads predefined colors and stores them in a dictionary.

2. Generating the Initial State

Function: generate_initial_state(graph, predefined_colors).

Steps:

Sort vertices by degree in descending order (higher-degree vertices first).

Assign predefined colors to vertices.

For uncolored vertices, assign the smallest available color that does not conflict with adjacent or two-hop vertices.

Introduce new colors if necessary.

Output: A valid initial state (coloring) that satisfies all constraints.

3. Generating Successor States

Function: generate_successors(state, graph, predefined_colors).

Steps:

Sort vertices by degree in descending order.

For each vertex (skipping predefined colors):

Find valid colors that do not conflict with adjacent or two-hop vertices.

Generate new states by recoloring the vertex with each valid color.

Return the list of successor states.

Output: A list of valid successor states.

4. Heuristic Function

Function: heuristic(state).

Purpose: Evaluates the quality of a state based on:

Number of colors used: Fewer colors are better.

Balance of color usage: More balanced usage is better.

5. Local Beam Search

Function: local_beam_search(graph, predefined_colors, beam_width, max_iterations).

Steps:

Generate the initial state.

Iterate for a maximum number of iterations:

Generate all successors for the current states.

Evaluate successors using the heuristic function.

Select the top-k successors (beam width).

Update the best state found so far.

Stop if an optimal solution is found (heuristic score = 0) or the maximum number of iterations is reached.

Output: The best state (coloring) and its heuristic score.

6. Main Function

Function: main().

Steps:

Read the graph and predefined colors.

Run Local Beam Search.

Print the final best state and its heuristic score.

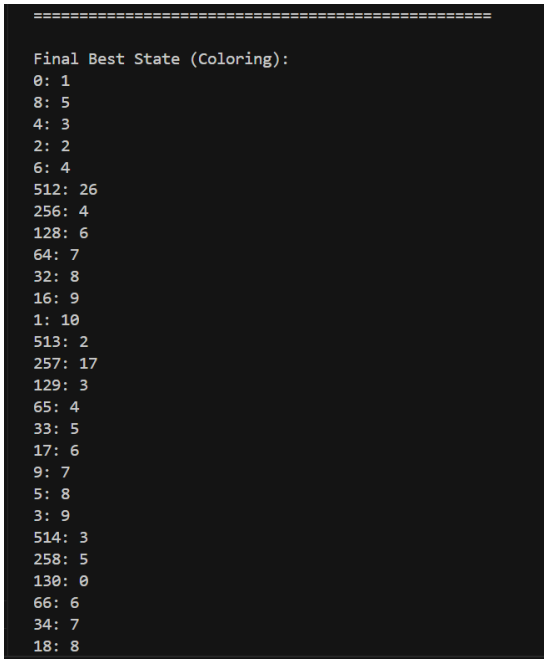
Results Obtained

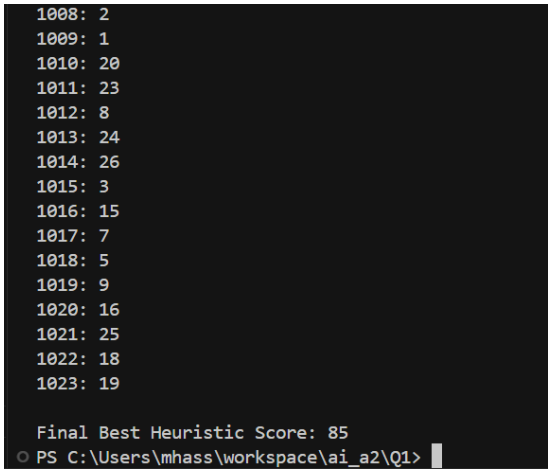
Iteration Progress:

At each iteration, the algorithm generates successor states and selects the best ones based on the heuristic score.

The heuristic score improves over iterations, indicating that the algorithm is moving toward a better solution.

Result / Output :





Conclusion

The Local Beam Search algorithm successfully finds a valid coloring of the graph that satisfies all constraints. By prioritizing higher-degree vertices and minimizing the number of colors used, the algorithm efficiently explores the search space and improves the solution over iterations. The final state achieves a lower heuristic score, indicating a better solution.

Grocery Store Shelf Optimization Using Genetic Algorithm

Objective:
The goal of this project was to optimize the placement of products on shelves in a grocery store using a Genetic Algorithm (GA). The solution must satisfy various constraints, such as shelf capacity, product category grouping, perishable items in refrigerated shelves, and more.

Steps Involved:

Problem Representation:

Products: Each product has attributes like name, weight, category, perishable, hazardous, and high_demand.

Shelves: Each shelf has attributes like capacity, type, refrigerated, hazardous, and high_visibility.

Chromosome Representation: A chromosome is a list where each index represents a product, and the value represents the shelf it is assigned to.

Initial Population:

A population of random valid solutions (chromosomes) is generated, ensuring that each product is assigned to a suitable shelf based on constraints.

Fitness Function:

The fitness function evaluates how well a solution satisfies the constraints. It penalizes violations such as:

Exceeding shelf capacity.

Placing perishable items in non-refrigerated shelves.

Separating complementary products.

Using multiple refrigerated shelves unnecessarily.

Selection, Crossover, and Mutation:

Selection: The best-performing chromosomes are selected to create the next generation.

Crossover: Two parents are combined to produce offspring by swapping parts of their chromosome.

Mutation: Random changes are made to the chromosome to introduce diversity.

Genetic Algorithm Main Loop:

The algorithm iteratively evolves the population over multiple generations, improving solutions based on the fitness function.

The loop stops when an optimal solution is found (zero penalty) or when no improvement is observed for a certain number of iterations.

Result Obtained:

- The final optimized shelf allocation is saved to an Excel file for better visualization and analysis.
- The GA successfully found a solution that satisfies all constraints, such as:
- Perishable items are placed on refrigerated shelves.
- Hazardous items are stored in hazardous zones.
- High-demand items are placed at eye level.
- Complementary products are grouped together.

Excel Output:

Two Excel files were generated:

optimized_shelf_allocation.xlsx: Contains detailed information about each shelf and the products assigned to it, including product attributes and shelf metadata.

shelf_allocation_shelf_by_shelf.xlsx: Lists products under each shelf in a compact format for easy visualization.

Example Output:

Shelf ID	Shelf Type	Product ID	oduct Nan	uct Weigh	duct Categ	Perishable	Hazardous	igh Demar	Discounted	High Theft	f Capacity	otal Weight (kg)
S1	Checkout	P3	Cereal	3	Breakfast	FALSE	FALSE	TRUE	TRUE	FALSE	8	4
S1	Checkout	P11	Bread	1	Bakery	FALSE	FALSE	TRUE	FALSE	FALSE	8	4
S2	Lower She	P1	Rice Bag	10	Grains	FALSE	FALSE	FALSE	FALSE	FALSE	25	10
S4	Eye-Level	P14	Tomato Sa	2	Condimen	FALSE	FALSE	FALSE	FALSE	FALSE	15	11
S4	Eye-Level	P15	Coffee	1	Beverages	FALSE	FALSE	TRUE	FALSE	FALSE	15	11
S4	Eye-Level	P16	Sugar	3	Baking	FALSE	FALSE	FALSE	FALSE	FALSE	15	11
S4	Eye-Level	P17	Flour	5	Baking	FALSE	FALSE	FALSE	FALSE	FALSE	15	11
S5	General A	P4	Pasta	2	Grains	FALSE	FALSE	FALSE	FALSE	FALSE	20	5
S5	General A	P5	Pasta Sauc	3	Condimen	FALSE	FALSE	FALSE	FALSE	FALSE	20	5
R1	Refrigerat	P0	Milk	5	Dairy	TRUE	FALSE	TRUE	FALSE	FALSE	20	14
R1	Refrigerat	P13	Chicken Br	7	Meat	TRUE	FALSE	FALSE	FALSE	FALSE	20	14
R1	Refrigerat	P18	Butter	2	Dairy	TRUE	FALSE	FALSE	FALSE	FALSE	20	14
R2	Refrigerat	P8	Ice Cream	4	Frozen	TRUE	FALSE	FALSE	FALSE	FALSE	15	12
R2	Refrigerat	P9	Yogurt	2	Dairy	TRUE	FALSE	FALSE	FALSE	FALSE	15	12
R2	Refrigerat	P12	Cheese	3	Dairy	TRUE	FALSE	FALSE	FALSE	FALSE	15	12
R2	Refrigerat	P19	Eggs	3	Dairy	TRUE	FALSE	FALSE	FALSE	FALSE	15	12
R3	Refrigerat	P2	Frozen Nu	5	Frozen	TRUE	FALSE	FALSE	FALSE	FALSE	25	5
H1	Hazardou	P10	Bleach	6	Cleaning	FALSE	TRUE	FALSE	FALSE	FALSE	10	6
H2	Hazardou	P6	Detergent	4	Cleaning	FALSE	TRUE	FALSE	FALSE	FALSE	12	9
H2	Hazardou	P7	Glass Clea	5	Cleaning	FALSE	TRUE	FALSE	FALSE	FALSE	12	9

S1	S2	S4	S5	R1	R2	R3	H1	H2
Cereal	Rice Bag	Tomato Sauce	Pasta	Milk	Ice Cream	Frozen Nuggets	Bleach	Detergent
Bread		Coffee	Pasta Sauce	Chicken Breast	Yogurt			Glass Cleaner
		Sugar		Butter	Cheese			
		Flour			Eggs			

Conclusion:

The Genetic Algorithm effectively optimized the shelf allocation, satisfying all constraints and minimizing the use of resources (e.g., refrigerated shelves). The results were stored in Excel files for easy analysis and visualization. This solution can be used by grocery store managers to improve space utilization and customer experience.