Trekking Route Planning for Nepali Mountain Trails based on Time, Distance and Difficulty

Rohit Kumar Sah
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, Illinois, USA
rohsah@siue.edu

Dipesh Duwal
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, Illinois, USA
dduwal@siue.edu

***Abstract*—Trekking in Nepal's mountainous terrain presents numerous significant challenges. It requires choosing and optimizing routes based on one's time availability, trail distance, key locations and trail difficulty. This report proposes an intelligent route-planning system that efficiently generates trekking trails with key locations while minimizing difficulty within a given time frame. The problem is modeled as a variant of the 0/1 Knapsack problem, where the objective is to select an optimal subset of locations while considering constraints on trekking difficulty and available days. To solve this, we implement and evaluate three algorithmic approaches: Greedy Best-First Search (GBFS), Divide and Conquer (Recursive Trail Partitioning), and Dynamic Programming (0/1 Knapsack DP). Each algorithm is assessed based on runtime efficiency, memory consumption, and solution quality under different constraints, including minimizing altitude gain (to prevent altitude sickness) and maximizing location importance. This project contributes to enhancing trekking experiences by providing data-driven, efficient, and customizable route recommendations for hikers navigating Nepal's trails.**

I. Big Problem

Nepal, a country known for its breathtaking mountain landscapes, has numerous trekking trails. Each of these trekking routes has unique challenges regarding difficulty, distance, and time factor. Traditionally, route planning is heavily dependent on subjective recommendations and experience-based navigation, which often lacks a systemic approach to optimize the trekking experience. So, it is quite challenging to select an optimal trekking route that minimizes difficulty while maximizing the number of key locations visited. To address this issue, we are developing a data-driven trekking route planning system that suggests the most efficient and feasible route that can be travelled within a given time duration from a starting point to a destination point. The system aims to integrate geospatial data to develop a weighted graph representation of trekking trails, nodes as significant locations, and edges as bearers of distance, difficulty, and importance of location.

Using algorithmic techniques such as Greedy selection (GBFS), Divide and Conquer (Recursive Trail Partition), and Dynamic Programming (0/1 Knapsack), the system generates optimized trekking routes based on the constraints of the user. The main objective is to ensure that recommended routes are within the available time frame while minimizing difficulty and maximizing the important location. Moreover, the system will be tested for scalability, adaptability, and real-world performance analysis based on comparisons between generated routes and popular trekking routes. Through this approach, the proposed system will offer enhanced trekking experiences by making Nepal's mountain trails accessible and enjoyable for trekkers of all levels with varied time availability.

II. Dataset

This dataset contains geographic information about trekking trails in the Sindhupalchok district of Nepal, focusing on trails and pathways around the Barhabise area. The data includes crucial parameters such as coordinates, elevation, road classifications, and location importance metrics that are crucial for any trekking planning decisions.

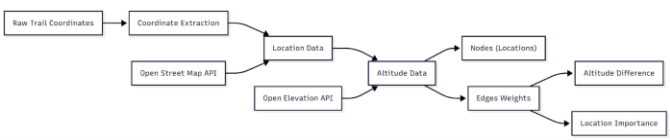

Figure 1: Flow Chart showing data processing steps

The project uses two primary datasets: (1) Geo JSON data having coordinates of major trekking trails of Gaurishankar Conservation Area, sourced from the National Trust for Nature Conservation, and (2) additional data from OpenStreetMap including altitude, location names, and attractions based on the trail's coordinates. The initial dataset lacks critical information about trail locations and terrain difficulty. This data will be converted into nodes where the vertices represent connected ways and the edge weight, we are considering Altitude Difference and Location Importance. For any result, we are trying to minimize altitude difference (for preventing altitude sickness) and maximize location importance.

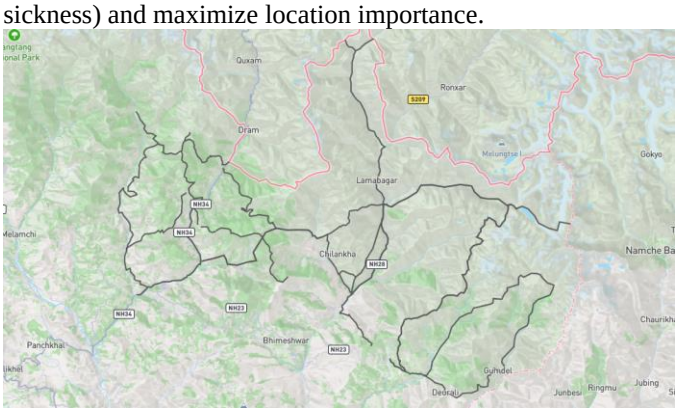

Figure 2: A map view plot of the trekking trails from

Gaurishankar Conservation Area based on trails coordinates (created using mapshaper).

Nodes: Each node represents a geographic location with attributes including precise coordinates, name, address information, and altitude.
Edges: Connections between geographic points, primarily following existing trails in the dataset and roads.
Edge Weights: Calculated primarily based on altitude differences between connected points (Trekking towards increasing altitude is always difficult), representing trail difficulty and location importance difference between points (This metric helps identify attractive points along potential trails).

Feature Name	Feature Description
latitude	Latitude of the location.
longitude	Longitude of the location.
location_name	Name of the location. (Node Identifier)
location_addresstype	Address Type of the location.
location_boundingbox	Bounding Box Coordinates (Area of the specific node)
location_class	Class of the location
location_display_name	Detailed address of the location
location_importance	Location Importance based on OSM (Algorithms should maximize this.)
location_osm_id	OSM Id for the location
location_place_id	OSM Place Id for the location
location_place_rank	OSM Place Rank for the location
address_city_district	Location Address District
address_county	Location Address County
address_municipality	Location Address Municipality
address_road	Location Address Road
address_village	Location Address Village
address_hamlet	Location Address Hamlet
altitude	Location Address Altitude (The difference between current and next node should be minimized.)

Table 1: Attribute Description of the finalized dataset after processing.

	Latitude	**Longitude**	**Location Importance**	**Location Place Rank**	**Altitude**
count	381	381	381	381	381
mean	27.82	86.11	0.08	24	2456.86
std	0.12	0.24	0.05	4	1174.25
min	27.57	85.75	0	18	684
25%	27.75	85.87	0.05	19	1548
50%	27.82	86.16	0.05	26	2260
75%	27.89	86.32	0.15	26	3424
max	28.13	86.55	0.16	30	5726

Table 2: Statistics of the final dataset after processing

A. Generated Datasets (For Experiment 1 & 2)

For experiments 1 and 2, to stress test the algorithms, we are going to generate a dataset with random numbers based on the requirements. For the first experiment since we are evaluating its scalability the same query parameters will be tested on networks of increasing size. So, different networks from size 10, 50 … 10000 vertices will be generated for testing. Similarly, for experiment 2, a large dataset with random edge weights is generated but queried with different input days limits.

III. Methodology

For solving the optimal trekking route problem, our approach is based on three algorithms: Greedy Best-First Search (GBFS), Recursive Trail Partitioning (Divide and Conquer), and 0/1 Knapsack (Dynamic Programming). These algorithms address different constraints, such as minimizing difficulty, and ensuring an efficient path within the given time duration. By leveraging geographic and elevation data, our methodology systematically processes trekking trails to generate feasible and optimized routes. The following section outlines the step-by-

step execution of these algorithms, including a structured data processing plan, block diagram showing program flow and an intuitive explanation of each approach.

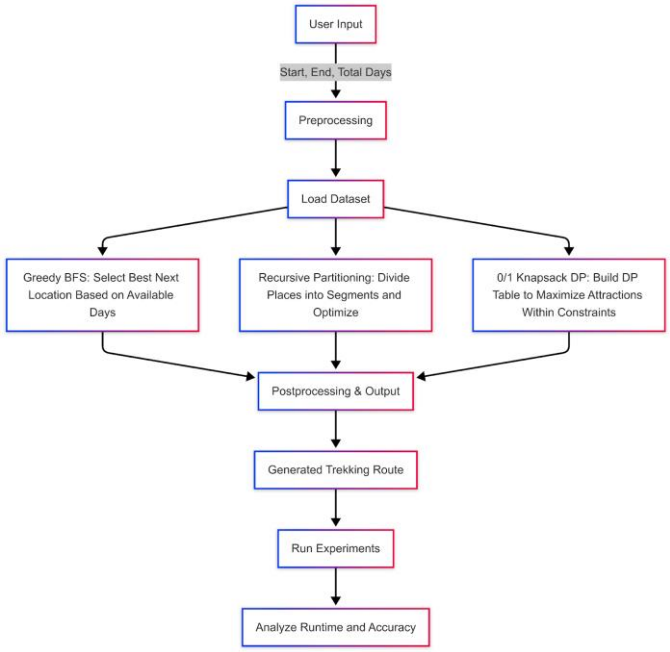

Figure 3: Block diagram of the program

A. Preprocessing

The first and foremost step in creating a trekking route planning system is the pre-processing step. It involves transforming the raw dataset into a structured format suitable for algorithmic processing. The dataset contains information such as location names, coordinates, altitude, road classifications, and location importance. Each location is treated as a node with attributes like name, latitude, longitude, importance, and altitude. The difficulty of a location is determined based on its altitude, with higher altitudes contributing to greater difficulty. The data is then used to calculate scores for the connections between locations, factoring in distance, altitude differences, and location importance. This processed data forms the basis for implementation of algorithms in upcoming steps.

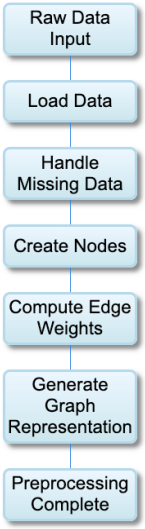

Figure 4: Data flow chart

B. Greedy Algorithm Approach: Best-First Search (GBFS) for Trail Selection

The greedy approach always picks the best-looking option at the moment without considering future consequences. Greedy best-first search makes a decision at every step by starting at a beginning location and iteratively choosing the next best place given by the evaluation equation. It prioritizes places that maximize location importance while minimizing travel distance and difficulty. The algorithm always chooses the place with the highest score. However, this greedy nature also means that it does not guarantee to find the most optimal path in the long run. The equation used for selection:

$$Score(p) = \frac{\text{Location Importance}(p)}{\text{Distance}(current, p) + 1} \times \frac{1}{\text{Difficulty}(p)}\}$$

Time Complexity: O(nlogn) (due to sorting locations at each step).
Space Complexity: O(n) (storing visited places).

Algorithm 1 Greedy Best-First Search (GBFS) for Trail Selection

```
Initialize trail with the start location
Set current_place to start and remaining_days to total available days
while remaining_days > 0 and current_place ≠ end do
    Find unvisited places excluding the end location
    if no unvisited places remain then
        Break
    end if
    Select the nearest unvisited place
    if it can be reached within available days then
        Add the place to the trail
        Update current_place and decrement remaining_days
    else
        Break
    end if
end while
if end location is not in the trail then
    Append end location to the trail
end if
return final trail
```

Figure 5: Pseudocode showing Greedy Best-first Search.

C. Divide and Conquer Approach - Recursive Trail Partitioning

Recursive Trail Partitioning (Divide and Conquer) works in a different way. Instead of selecting one place at a time, this approach splits the problem into smaller subproblems. This algorithm divides all intermediate places into two halves, recursively finds the best segment in each half, and then merges the results into a single sorted segment based on predetermined factors like altitude, time and location importance. Finally, it returns the best set of places that fit within the available days. It makes sure that each segment is balanced and collectively forms a feasible trekking path. It also ensures efficiency while processing large datasets because his problem follows the recurrence relation.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Time Complexity: O(nlogn)
Space Complexity: O(n)

Algorithm 2 Divide and Conquer - Recursive Trail Partitioning

```
function FIND_BEST_SEGMENT(places, days, start_place)
    if places are empty or days = 0 then
        return []
    end if
    if only one place remains then
        return that place
    end if
    Split places into left and right halves
    Recursively find best segment in the left half
    Use last place from left segment as start for right segment
    Recursively find best segment in the right half
    Combine both segments
    Sort combined segment based on predefined scoring criteria
    Select the best 'days' number of places
    return final segment
end function
```

Figure 6: Pseudocode showing Divide and Conquer - Recursive Trail Partitioning.

D. Dynamic Programming Approach (0/1 Knapsack)

Unlike the above approaches, Dynamic Programming approach tries all the possible combinations of locations and days while storing the intermediate results in order to avoid redundant calculations. It may be an exhaustive process, yet it is an optimized approach. In this approach, a Dynamic Programming (DP) table is created where each entry dp[i][j] represents the best possible score when selecting from the first i places within j days. The algorithm makes binary choices at each step—either including a place if it fits within the available time or excluding it and inheriting the previous best score. The decision is made using the state transition formula:

$$dp[i][j] = \max\big(dp[i-1][j], dp[i-1][j - \text{travel\_days}(i)] + Score(i)\big)$$

where score(i) represents the location importance and difficulty level. By filling the DP table iteratively, the algorithm finds the most optimal route. Although highly effective, this approach requires more memory and processing time compared to the other two approaches.

Time Complexity: O(n·d) where n is the number of places and d is the number of days.
Space Complexity: O(n·d) (for storing DP table).

Algorithm 3 Dynamic Programming - 0/1 Knapsack for Trail Selection

```
Initialize DP table dp[i][j] for i places and j days
for each place i do
    for each day j from 1 to total_days do
        Option 1: Exclude current place (inherit previous best)
        for each possible day allocation k do
            Identify previous best trail and last visited place
            if travel is feasible within k days then
                Calculate score based on importance and altitude
                Update DP table if new score is better
            end if
        end for
    end for
end for
Extract best trail from DP table
if end location is not included then
    Append end location to the trail
end if
return final optimized trail
```

Figure 7: Pseudocode showing Dynamic Programming - 0/1 Knapsack for Trail Selection.

IV. Experiment Design

For all the experiments listed below, we'll be measuring runtime in seconds, and solution quality (depending on the average altitude and average location importance of the recommended trail). The runtime of the algorithms will be tested on home.cs.siue.edu to ensure consistent and reproducible results.

A. Experiment 1 (Testing Scalability)

Experiment 1 assesses the scalability of the algorithm across different network sizes. For this experiment, datasets of increasing sizes will be randomly generated, as explained in the dataset section. The dataset sizes (10, 50, 100, 500, … 1000) will be tested against the algorithm, and their runtime will be measured. This experiment examines how increasing network size affects all three algorithms, with the results plotted against time. To ensure a fair comparison, network size is the only variable, while all other parameters remain constant throughout the experiment.

B. Experiment 2 (Testing Adaptability)

Experiment 2 evaluates how well the algorithms handle different constraints, testing their robustness and adaptability. We will vary factors such as the number of days (3, 7, 14, and 30 days), location importance, and difficulty thresholds. To assess performance, we will measure satisfaction rates, which refer to both successfully identifying trails and selecting the best ones by maximizing importance while minimizing difficulty. The experiment will analyze how often constraints are violated, the percentage of feasible solutions under different conditions, and how well the algorithms balance competing constraints. Throughout these tests, the dataset will remain the

same, while different input parameters will be applied to evaluate the algorithms' performance.

C. *Experiment 3 (Testing Real World Performance)*

Experiment 3 evaluates the performance of the algorithm on a real-world dataset. All three algorithms are tested on the processed dataset, as described in the dataset section, and are analyzed based on runtime and memory usage. The results will be plotted, showing how runtime changes as the number of days increases with different input parameters.

Another part of this experiment, which cannot be measured numerically, involves testing popular trekking trails in the area. We will compare the algorithm's suggested routes to actual paths taken by trekkers. By setting a start and destination point, we can assess whether the algorithm's routes align with real-world trekking routes. Additionally, we will compare how well the algorithm covers key attractions compared to traditional trekking paths.

V. GANTT CHART

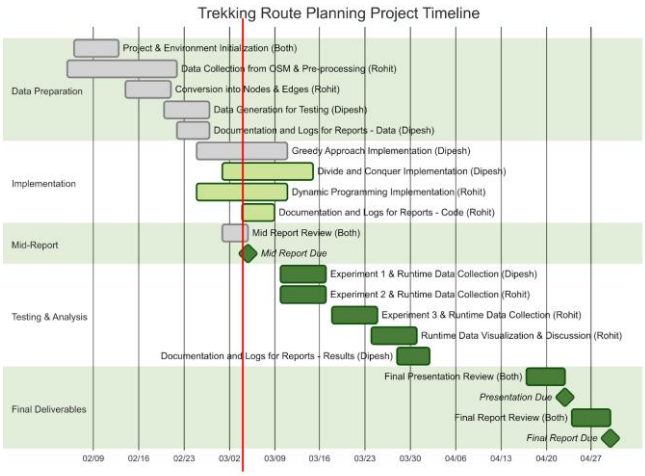

Figure 8: Gantt chart showing project timeline (completed tasks are highlighted with grey color, active tasks are highlighted with light green whereas pending tasks are highlighted with green color)

VI. CONCLUSION

For the project we are mostly good as we are on time as per the Gantt chart shown above.

The initial dataset we had was coordinates of a trail in the Gaurishanker Area. For each coordinate, the location data is collected from Open Street Map and altitude data is collected from the Open Elevation API. Then these locations are clustered together to form a single node or location. The altitude difference and location importance difference can be considered as edge weights or deciding factor for a specific route.

The three experiments that are being conducted are for each testing scalability, adaptability and real-world performance. For each experiment a specific parameter of the algorithm was varied keeping all other parameters constant. This helps us analyze the impact of a specific parameter on the overall algorithm.

REFERENCES

[1] Cacchiani, Valentina, et al. “Knapsack Problems — an Overview of Recent Advances. Part I: Single Knapsack Problems.” Computers & Operations Research, vol. 143, Feb. 2022, p. 105692. https://doi.org/10.1016/j.cor.2021.105692.

[2] NTNC Geoportal. geoportal.ntnc.org.np. Accessed 4 Mar. 2025.

[3] Sniedovich, Moshe. “Dynamic Programming Algorithms for the Knapsack Problem.” ACM SIGAPL APL Quote Quad, vol. 24, no. 3, Mar. 1994, pp. 18–21. https://doi.org/10.1145/181983.181988.