An HTN Planning Approach for Ambulance Victim-Rescue

Louis Hähle, Francesco Carlucci

March 12, 2025

1 Introduction

Hierarchical Task Network (HTN) planning is an approach for solving complex problems by decomposing high-level goals into smaller, more manageable subtasks. This decomposition allows for dynamic adaptation to changing conditions, making HTN well-suited for state-dependent, multi-step problems. Pyhop is a Python-based HTN planning framework that models tasks through methods and actions through operators. Methods recursively decompose tasks into subtasks, while operators define the actions that modify the system state.

In this paper, we apply an HTN-based solution (implemented with Pyhop) to the problem of ambulance allocation and victim rescue. The primary objective is to dynamically assign ambulances to victims based on proximity and severity, determine the shortest path to the victim, transport them to the nearest hospital, and update the system state accordingly.

This problem is highly structured and goal-driven, making HTN planning a natural fit. The overall process can be broken down hierarchically into distinct steps: assigning a victim, moving the ambulance, loading the victim, navigating to the hospital, unloading the victim, and releasing the ambulance. Pyhop's capacity to handle recursive task decomposition and state updates makes it especially well-suited for this domain.

The paper presents the implementation details, highlights the key components (state, operators, methods), and demonstrates the system's effectiveness through illustrative scenarios.

2 Problem Description

2.1 General setting

The environment is represented by an undirected graph $G = (V, E)$, where each vertex $v \in V$ is a location and each edge $(u, v) \in E$ indicates a path for traveling between locations. Certain locations are designated as hospitals, while ambulances and victims can also be present at any location. Each victim has a severity level (1–10), reflecting the seriousness of their condition, and each ambulance has a capacity indicating the maximum severity it can handle.

The graph structure, the location of hospitals, and the maximum capacity of each ambulance are fixed elements of the problem. In contrast, positions of victims and ambulances are dynamic and can change over time.

The goal is to devise a plan that uses the available ambulances to transport all victims to a hospital, respecting the severity-handling capacity of the ambulances and the connectivity constraints of the graph. The closest available ambulance capable of handling a victim's severity should be assigned, and the ambulance should follow the shortest possible path to the hospital. Victims with a severity level that exceeds a threshold of 6 need to be provided first aid before being transported to the hospital.

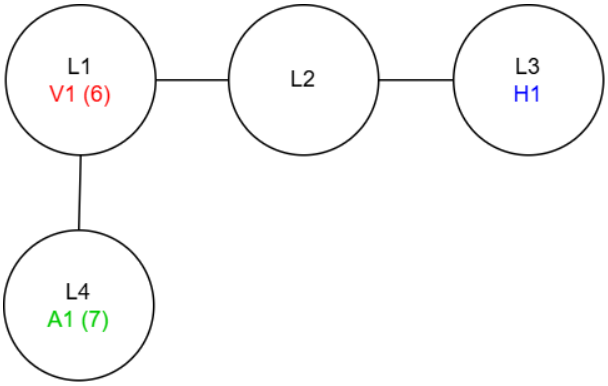

Figure 1: Simple problem example.

2.2 Example Scenario

Figure 1 illustrates a basic example of the problem. There are four locations:

- **L1** contains a victim labeled **V1** with a severity level of **6** that requires medical attention.
- **L2** is an empty location that serves as a connection point between other locations.
- **L3** contains a hospital labeled **H1**, which is the only hospital in this scenario.
- **L4** contains an ambulance labeled **A1** with a capacity of **7**, meaning it can handle the severity level of **V1**.

The goal is to transport all victims to a hospital. In this example, the solution would involve ambulance **A1** picking up victim **V1** at **L1** and transporting them to the hospital at **L3**. Since the severity of the victim does not exceed 6, first aid is not required.

3 Expert Knowledge Modeling

The expert knowledge modeling follows a structured approach to represent decision-making and automation processes. In our case the main goal is to transport all the victims to an hospital to be treated. Below is a detailed specification of the Expert Knowledge Modeling of the project.

3.1 Main task

treat_all_victims: this is the ultimate goal. If all victims are treated, the system stops, otherwise, it executes:

- **assign_goals**: Assigns ambulances to waiting victims.
- **do_step**: Moves ambulances and handles task completion.
- Calls itself until all victims are treated.

3.2 Methods

- **assign_goals**: Finds a waiting victim and assigns them to an available ambulance.
- **do_step**: Moves ambulances simultaneously along their path and handles what to do when they reach their destination.

- **handle_goal_completion**: Decides whether to load a victim or unload them at the hospital.
- **first_aid_if_necessary**: Checks if a victim needs first aid or not.

3.3 Operators

- **assign_victim_op**: Assigns a victim to an ambulance.
- **move_ambulance_op**: Moves an ambulance from one location to another.
- **load_victim_op**: Loads a victim into the ambulance.
- **unload_victim_op**: Unloads a victim at the hospital.
- **provide_first_aid_op**: Provides first aid to a victim.

3.4 Visual Decomposition

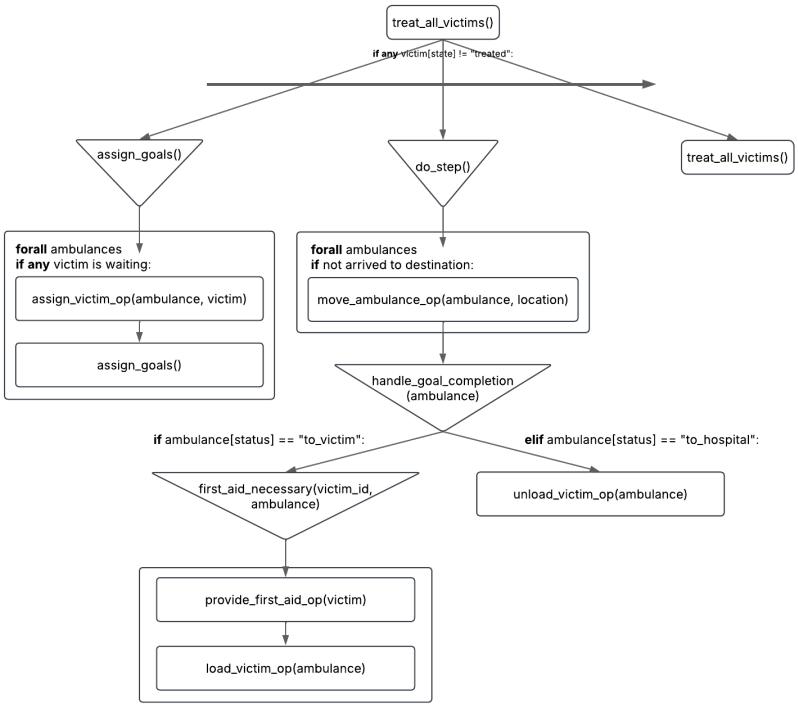

Figure 2: Graphical scheme for modeling expert knowledge

4 Implementation

4.1 State Representation

In our implementation the state is represented as an object (using `pyhop.State`) that contains information about the environment, including ambulances, victims, hospitals, coordinates, and connections. Each component (except for the graph) is stored as a dictionary and is identified by a unique key. Below is a detailed description of how each component is structured within the state:

- **Ambulances:**
  - **location**: The current location of the ambulance.
  - **capacity**: The maximum severity level of victims the ambulance can handle.
  - **path**: A list representing the historical path taken by the ambulance.
  - **state**: The current state of the ambulance, such as "available", "to_victim", or "to_hospital".
  - **current_path**: A list of locations representing the planned path the ambulance will follow in this state.
  - **victim**: The ID of the victim currently assigned to the ambulance.
  - **hospital**: The ID of the hospital the ambulance is assigned to.
- **Victims:**
  - **location**: The current location of the victim.
  - **severity**: The severity level of the victim's condition.
  - **first_aid_done**: A boolean indicating whether first aid has been provided.
  - **state**: The current state of the victim, such as "waiting", "ambulance_assigned", or "treated".
- **Hospitals:** Each hospital has just the **location** attribute that defines the location of the hospital.
- **Coordinates:** Each location is mapped to its X and Y coordinates. This allows the system to calculate distances between locations using the Euclidean distance formula.
- **Connections:** Each location is mapped to a list of directly connected locations. This represents the map's topology and is used to create the graph for pathfinding.
- **Graph:** The graph is created using the create_graph function, which uses the coordinates and connections to build the graph. It is then stored and used for computing shortest paths.

4.2 Operators

- `assign_victim_op(state, ambulance, victim)`: Assigns a waiting victim to an available ambulance if the ambulance has sufficient capacity.
  - **Preconditions** :
    - The victim's state must be "waiting".
    - The ambulance's state must be "available".
    - The victim's severity must be less than or equal to the ambulance's capacity.
  - **Effects on the State** :
    - The ambulance's state is updated to "to_victim".
    - The victim's state is updated to "ambulance_assigned".
    - The ambulance's victim attribute is set to the assigned victim.
    - The ambulance's current_path is updated to the shortest path from the ambulance's current location to the victim's location.
- `load_victim_op(state, ambulance)`: Loads a victim into the ambulance if the ambulance has reached the victim's location.

  - **Preconditions**:
    * The ambulance must be at the same location as the victim.
    * The ambulance's state must be "to_victim".
    * The victim's severity must be less than or equal to the ambulance's capacity.
  - **Effects on the State**:
    * The ambulance's state is updated to "to_hospital".
    * The ambulance's hospital attribute is set to the nearest hospital.
    * The victim's location is updated to the ambulance (indicating they are being transported).
    * The ambulance's current_path is updated to the shortest path from the victim's location to the hospital.

- `unload_victim_op(state, ambulance)`: Unloads a victim at the hospital if the ambulance has reached the hospital.
  - **Preconditions**:
    * The ambulance must be at the same location as the hospital.
    * The victim must be in the ambulance.
  - **Effects on the State**:
    * The victim's state is updated to "treated".
    * The victim's location is updated to the hospital.
    * The ambulance's state is reset to "available".
    * The ambulance's victim and hospital attributes are set to None.

- `move_ambulance_op(state, ambulance, loc)`: Moves an ambulance from its current location to the next location in its current_path.
  - **Preconditions**: The next location must be connected to the ambulance's current location.
  - **Effects on the State**:
    * The ambulance's location is updated to the next location.
    * The next location is appended to the ambulance's path (historical record).
    * The next location is removed from the ambulance's current_path.

- `provide_first_aid_op(state, victim)`: Provides first aid to a victim.
  - **Effects on the State**: The victim's first_aid_done attribute is set to True.

4.3 Methods

- `assign_goals(state)`: Assigns an available ambulance to any waiting victim.
  - **Logic**:
    * Checks if there are any waiting victims.
    * If no victims are waiting, returns an empty list (nothing to do).
    * Otherwise, iterates through all ambulances to find an available one.
    * Uses `assign_victim` to find the nearest victim for the ambulance.
    * Returns a list of subtasks: `assign_victim_op, assign_goals`.
  - **Returns**: A list of subtasks or an empty list.

- `first_aid_if_necessary(state, victim, ambulance)`: Provides first aid to a victim if their severity is high (≥ 7) and the ambulance is present.
  - **Logic**:
    - Checks if the victim's severity is ≥ 7, first aid has not been provided, and the ambulance is at the victim's location.
    - Returns the operator `provide_first_aid_op` if conditions are met.
  - **Returns**: A list containing the operator or an empty list.
- `do_step(state)`: Moves each ambulance step-by-step along its current path and handles goal completion when the path is finished.
  - **Logic**:
    - Iterates through all ambulances.
    - If the ambulance has a path, adds a `move_ambulance_op` subtask to move to the next location.
    - If the path is finished and the ambulance is en route to a victim or hospital, calls `handle_goal_completion` to determine the next steps.
  - **Returns**: A list of subtasks (move operations and goal completion tasks) or an empty list.
- `handle_goal_completion(state, ambulance)`: Handles the completion of a goal when an ambulance arrives at a victim or hospital.
  - **Logic**:
    - If the ambulance is en route to a victim:
      - Checks if first aid is necessary using `first_aid_if_necessary`.
      - Adds subtasks for first aid and loading the victim.
    - If the ambulance is en route to a hospital:
      - Adds a subtask for unloading the victim.
  - **Returns**: A list of subtasks.
- `treat_all_victims(state)`: Continues until all victims are in the `"treated"` state.
  - **Logic**:
    - Checks if all victims are treated.
    - If all victims are treated, returns an empty list (task completed).
    - Otherwise, returns a list of subtasks:
      - `assign_goals, do_step, treat_all_victims`
  - **Returns**: A list of subtasks or an empty list.

4.4 Helper Functions

We used some auxiliary functions to support the system's core operations:

- `distance(c1, c2)`: Calculates the Euclidean distance between two points, enabling measurement of distances between locations such as ambulances, victims, and hospitals.
- `create_graph(state)`: Constructs a graph representation of the environment using NetworkX, where nodes represent locations and edges are weighted by their Euclidean distances. This graph serves as the foundation for pathfinding and decision-making.

- `shortest_path(state, start, goal)`: Implements Dijkstra's algorithm to compute the shortest path and its associated cost between two locations.
- `assign_hospital(state, victim)`: Find the nearest hospital to the given victim.
- `assign_victim(state, ambulance)`: Finds the nearest waiting victim that the ambulance can handle, given its capacity.

5 Examples

To demonstrate the functionality of the proposed solution, this section presents three example scenarios. Each example represents a different configuration of ambulances, victims, and hospitals within the environment.

5.1 First example

This is a basic example illustrated in Figure 3. Since the implementation assigns each ambulance to the closest treatable victim, ambulance **A1** first picks up victim **V1** at **L1** and transports them to the hospital at **L3**. After completing this task, **A1** picks up victim **V2** at **L2** and transports them to the hospital as well.

Note that the coordinates and distances are not explicitly shown here but are based on the relative distances depicted in the figure. Table 5.1 shows the exact steps taken during the solution process. In our implementation, ambulances move one step at a time, which is why the table records actions at discrete time steps. While no explicit time notion exists in the code, this representation helps improve clarity.

The table does not list the specific methods called during execution; instead, it summarizes the concrete actions performed. For each action shown in a cell, there exists a corresponding operator in the code responsible for executing that action (`move_ambulance_op` for moving an ambulance, for instance).

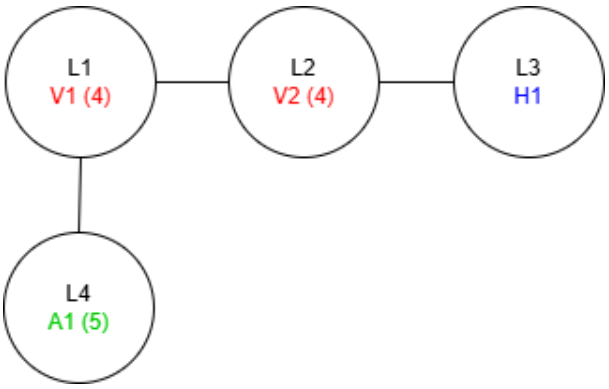

Figure 3: Initial state with four locations, two victims (V1 at L1, V2 at L2), one ambulance (A1 at L4), and one hospital (H1 at L3)..

5.2 Second example

A more sophisticated example is shown in Figure 4. Since there are two **ambulances** and three **victims**, the assignment of victims to ambulances becomes a key part of the problem. As described in the implementation section, each ambulance is assigned to the closest treatable victim, processed in order of ambulance index from lowest to highest.

t	A1
0	assign Victim V1
1	Move to L1
2	Load victim V1
3	Move to L2
4	Move to L3
5	Unload victim A1 to H1
6	Assign victim V2
7	Move to L2
8	Load victim V2
9	Move to L3
10	Unload victim A2 to H1

Table 1: Table showing the solution to Example 1. t indicates the time step, and the column for A1 details the steps taken by ambulance A1

In this case, ambulance **A1** is closest to both **V2** and **V3**, but since **V2** has the lower index, **A1** is assigned to **V2** first. Since **A2** is closest to **V1** but lacks sufficient capacity to treat a severity of **8**, **V3** becomes the next closest treatable victim and is therefore assigned to **A2**.

Once **A1** has finished providing first aid to **V2**, it becomes available again and is assigned to **V1**. Because **V1**'s severity exceeds **6**, first aid is provided before transporting the victim to the closest hospital, **H2**.

The implementation follows a greedy algorithm that assigns the closest treatable victim at each step without considering the overall optimal outcome. As shown in this example, this can lead to suboptimal results—for instance, the solution would have been faster if **A1** had provided first aid to the more severely injured victim **V3** first. The exact steps taken by the ambulances are shown in Table 5.2.

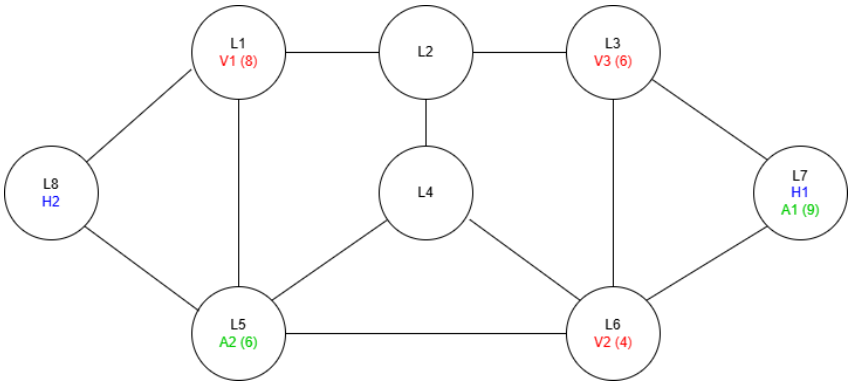

Figure 4: Initial state for Example 2 with eight locations, three victims (V1 at L1 with severity 8, V2 at L6 with severity 4, V3 at L3 with severity 6), two ambulances (A1 at L7 with capacity 9, A2 at L5 with capacity 6), and two hospitals (H1 at L7, H2 at L8).

5.3 Third example

A more complex example is illustrated in Figure 5. Since the graph is now more intricate, not all node positions directly correspond to the actual distances between the locations. To verify the exact distances, the accompanying code can be consulted. The graph is intended to provide a clearer overview of the problem structure rather than represent exact distances.

This scenario includes three ambulances, each starting at a hospital. As before, each am-

t	A1	A2
1	assign victim V2	assign victim V3
2	Move to L6	Move to L4
3	Load victim V2	Move to L2
4	Move to L7	Move to L3
5	Unload victim V2 at H1	Load victim V3
6	Assign victim V1	Move to L7
7	Move to L3	Unload victim V3 at H1
8	Move to L2	
9	Move to L1	
10	Provide first aid and load victim V1	
11	Move to L8	
12	Unload victim V1 at H2	

Table 2: Table showing the solution to Example 2. t indicates the time step, and the columns for A1 and A2 detail the steps taken by ambulances A1 and A2, respectively. Note that providing first aid occurs in the same time step as loading the victim.

bulance is assigned to the closest treatable victim, processed in order of ambulance index from lowest to highest.

- **A1** is assigned to the closest victim it can handle, which is **V5** at **L14**. After picking up **V5**, the ambulance transports them to the closest hospital, which is **H2** at its starting location.
- **A2** is closest to **V1** and **V2**, but their severity exceeds its capacity of **7**. Therefore, **A2** is assigned to **V3**, which it can handle. After loading **V3** at **L8**, it transports them to the closest hospital, **H2**.
- **A3** is assigned to the closest victim it can handle, which is **V4** at **L9**. After picking up **V4**, **A3** transports them to the starting hospital, **H1**, at **L4**. Since the only other victims with low enough severity for **A3** to handle have already been treated, **A3** remains inactive for the remainder of the scenario.

Once victims **V5** and **V4** are transported to the hospital, **A1** becomes available again and is assigned to **V1** at **L2**. Since **V1** and **V2** are both at the same location, **V1** is selected first because it has the lower index. Because their severity exceeds **6**, both victims receive first aid before being transported to the nearest hospital, **H3**, at **L7**. **A1** first provides first aid to **V1** and transports them to **H3**. After that, **A1** provides first aid to **V2** and follows the same route to **H3**.

Meanwhile, **A2** is assigned to the last remaining victim, **V6**, at **L13**. After picking up **V6**, **A2** transports them to the nearest hospital, **H3**, at **L7**. As before, the exact steps taken are shown in Table 3.

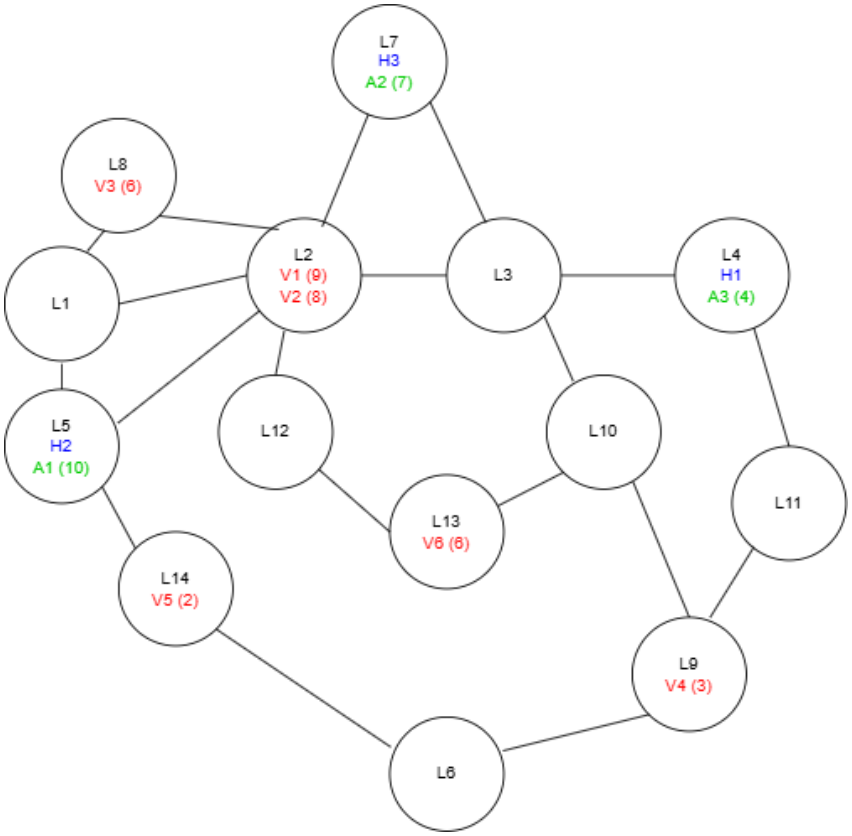

Figure 5: Initial state for Example 3 with fifteen locations, six victims (V1 at L2 with severity 9, V2 at L2 with severity 8, V3 at L8 with severity 6, V4 at L9 with severity 3, V5 at L14 with severity 2, and V6 at L13 with severity 6), three ambulances (A1 at L5 with capacity 10, A2 at L7 with capacity 7, and A3 at L4 with capacity 4), and three hospitals (H1 at L4, H2 at L5, and H3 at L7).

6 Conclusion

This paper presented an HTN-based approach using Pyhop to solve the problem of ambulance allocation and victim rescue. The proposed solution effectively assigned ambulances to victims based on proximity and capacity, successfully handling complex multi-agent coordination scenarios. The implementation includes a level of parallelism, allowing multiple ambulances to operate simultaneously, which better reflects how real-world emergency systems would function.

The greedy approach used in the implementation enabled fast, local decision-making but may lead to suboptimal global outcomes in certain cases.

Future improvements could focus on enhancing global optimality and introducing collaborative decision-making between ambulances.

Overall, the approach demonstrated effective task allocation and transport in emergency response scenarios. With further refinement, it could provide faster and more efficient victim rescue.

t	A1	A2	A3
0	Assign victim V5	Assign Victim V3	Assign Victim V4
1	Move to L14	Move to L2	Move to L11
2	Load victim V5	Move to L8	Move to L9
3	Move to L5	Load victim V3	Load victim V4
4	Unload victim V5	Move to L1	Move to L11
5	Assign victim V1	Move to L5	Move to L4
6	Move to L2	Unload victim V3	Unload victim V4
7	Provide first aid and load victim V1	Assign victim V6	
8	Move to L7	Move to L2	
9	Unload victim V1	Move to L12	
10	Assign victim V2	Move to L13	
11	Move to L2	Load victim V6	
12	Provide first aid and load victim V2	Move to L10	
13	Move to L7	Move to L3	
14	Unload victim V2	Move to L7	
15		Unload Victim V6	

Table 3: Table showing the solution to Example 3. t indicates the time step, and the columns for A1, A2, and A3 detail the steps taken by ambulances A1, A2, and A3, respectively. Note that providing first aid occurs in the same time step as loading the victim.