Optimizing Temporal Hypergraph Paths: A Time-Aware Hyperedge Hopping Approach

Ali Ahmadi Roghabadi
ali.ahmadi9@ut.ac.ir
University of Tehran

Abstract

Temporal hypergraphs provide a rich model for complex systems exhibiting dynamic, multi-way relationships, with applications spanning from social network analysis to biological pathways. Efficiently navigating these structures to find optimal paths presents unique computational challenges due to their inherent temporal constraints and higher-order connectivity. This paper **introduces and thoroughly details the Time-Aware Hyperedge Hopping (TAHH) algorithm**, a novel Dijkstra-like approach specifically designed to compute paths that achieve the earliest possible completion time within temporal hypergraphs. The TAHH algorithm strategically integrates temporal awareness into its relaxation process, enabling direct and agile path discovery without complex graph transformations. To evaluate its performance and validate its efficiency, TAHH is rigorously compared against a specialized Dijkstra variant, which optimizes for the "Min Earliest Entry Time" to hyperedges along a path. Our experimental results, conducted on large-scale synthetic temporal hypergraphs, demonstrate that both TAHH and the Min Earliest Entry Dijkstra variant provide highly efficient solutions, effectively handling the temporal complexities of hypergraph pathfinding. TAHH, in particular, emerges as a robust and scalable method for identifying optimal temporal paths, making it a valuable tool for real-world scenarios requiring swift and precise navigation through dynamic, high-dimensional temporal datasets.

Keywords: Temporal Hypergraphs, Pathfinding, Time-Aware Hyperedge Hopping, Dijkstra's Algorithm, Shortest Path, Temporal Networks, Algorithm Efficiency.

1 Introduction

In the study of complex systems, networks serve as fundamental tools for representing relationships and interactions among entities. Traditionally, these systems are modeled as graphs, where edges connect exactly two vertices, capturing pairwise interactions. However, many real-world phenomena involve interactions among more than two entities simultaneously. For instance, a scientific collaboration might involve several researchers on a single paper, a biological reaction could involve multiple molecules, or a meeting might include several participants. To accurately model such **higher-order interactions**, the concept of a **hypergraph** emerges as a powerful and more versatile generalization of a graph. A hypergraph $\mathcal{H} = (V, E)$ consists of a set of vertices V and a set of hyperedges E, where each hyperedge $e \in E$ is an arbitrary non-empty subset of vertices ($e \subseteq V$). Unlike standard edges, hyperedges can connect any number of vertices, providing a rich framework to describe complex, multi-way relationships.

The ability of hypergraphs to capture these higher-order structures makes them indispensable across a wide array of scientific and engineering disciplines. In **social network analysis**, hypergraphs model group collaborations, co-authorship networks, or participation in events. In **biology**, they are used to represent protein-protein interactions, metabolic pathways, or disease associations involving multiple genes. In **transportation and logistics**, hypergraphs can describe shared rides, public transport routes involving multiple stops, or supply chain stages with multiple participants. Furthermore, their applications extend to **database systems**, **data mining**, **computer vision**, and **security**, where understanding multi-faceted relationships is critical. Figure 1 illustrates a simple hypergraph structure.

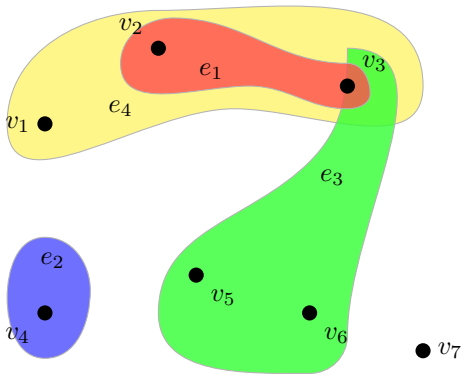

Figure 1: An example of a simple hypergraph. Vertices (e.g., v_1, v_2) are represented by black circles. Hyperedges (e.g., e_1, e_2) are represented by translucent shaded regions connecting multiple vertices. For instance, $e_1 = \{v_1, v_2, v_3\}$ and $e_4 = \{v_2, v_3\}$. Vertex v_7 is isolated.

2 Related Work

The problem of pathfinding has been extensively studied in graph theory and network science for decades, with classic algorithms like Dijkstra's [9] and Floyd-Warshall [11] providing foundational solutions for shortest paths in static networks. As real-world systems evolved, so did the complexity of their models, leading to advancements in understanding dynamic and higher-order interactions.

2.1 Pathfinding in Temporal Graphs

The inclusion of a temporal dimension to traditional graphs introduced new challenges and definitions for pathfinding. Temporal graphs, where edges are active only at specific time points or intervals, require paths to adhere to strict chronological order. Concepts such as "earliest arrival path" [4], "fastest path" [5], and "minimum duration path" [14] have been introduced. Algorithms for these problems often involve adaptations of Dijkstra's algorithm for time-dependent edge weights [8] or the construction of a large **Time-Expanded Graph (TEG)** [5]. While effective for pairwise interactions, these methods do not inherently address multi-way relationships.

2.2 Pathfinding and Analysis in Static Hypergraphs

To model interactions involving more than two entities, hypergraphs have gained prominence. Research in static hypergraphs has focused on generalizing graph-theoretic concepts, including measures of centrality [3], community detection [10], and structural properties [2]. Pathfinding in static hypergraphs, defined as a sequence of nodes where consecutive nodes are within the same hyperedge, has also been explored [1]. These "hyperpaths" aim to minimize steps or length in a static context [12]. Figure 2 provides a visual representation of pathfinding (routing) in a generic hypergraph structure. However, these studies abstract away the crucial temporal dimension, assuming constant availability of interactions.

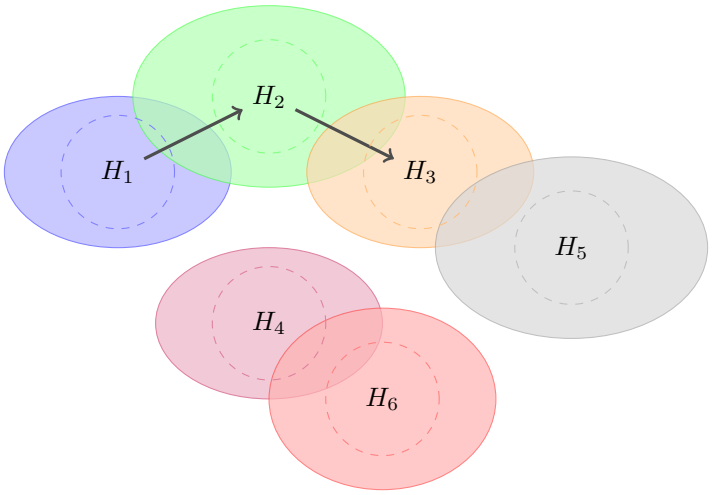

Figure 2: Conceptual demonstration of hyperedge-to-hyperedge routing in a hypergraph. Hyperedges are represented as colored, overlapping shapes, signifying shared vertices within their intersection areas. The black arrows illustrate an example path, traversing from H_1 to H_2 and then to H_3, where transitions occur through shared entities (implicit in the overlaps).

2.3 Prior Work on Temporal Hypergraphs

More recently, the intersection of higher-order interactions and temporality has spurred interest in **temporal hypergraphs**. Studies in this area have investigated the temporal dynamics of higher-order structures, including burstiness [6] and temporal-topological correlations [7]. Specific to pathfinding, some works have formalized concepts of paths in temporal hypergraphs. For instance, [13] defines a temporal hypergraph as a sequence of static hypergraphs indexed by time, and introduces "temporal paths" as ordered sequences of (node, time) pairs. Their work primarily focuses on analyzing the properties of fastest paths (shortest duration) and shortest paths (fewest steps) **between nodes** within various real-world temporal hypergraphs. They highlight the computational cost of analyzing large temporal networks and the challenges of characterizing system connectivity with higher-order interactions.

2.4 Research Gap and Our Contribution

While the aforementioned studies lay important groundwork, there remains a significant gap in efficiently identifying optimal paths **between hyperedges** in temporal hypergraphs, particularly with the objective of minimizing the absolute completion time of the path. Existing temporal pathfinding algorithms, including those in temporal hypergraphs, often operate on node-centric definitions or rely on computationally intensive transformations like the Time-Expanded Graph, which can become prohibitively large for dense or long-duration systems. Our work directly addresses this gap by introducing the **Time-Aware Hyperedge Hopping (TAHH) algorithm**. TAHH is a novel Dijkstra-like approach that directly operates on the temporal hypergraph structure, meticulously managing time validity and hyperedge-to-hyperedge transitions. This approach avoids the high overhead associated with explicit graph expansion, thereby offering a highly efficient and scalable solution for pathfinding aiming for the earliest final completion time. We provide a rigorous comparison of TAHH against a specialized Dijkstra variant optimizing for earliest entry time, demonstrating its effectiveness in navigating complex temporal hypergraph dynamics.

3 Temporal Hypergraph Model and Problem Formulation

To formally define the scope of our study and the problems we address, we first introduce the fundamental concepts of hypergraphs and extend them to the temporal domain. Subsequently, we formalize the notion of a path in such structures and delineate the specific optimization objectives.

3.1 Temporal Hypergraph Model

Building upon the definition of a static hypergraph (as introduced in Section 1), we define a **temporal hypergraph** as a triplet $\mathcal{H}_T = (V, E, \mathcal{T})$. Here, V and E retain their definitions from a static hypergraph. The new component, $\mathcal{T}$, is a function $\mathcal{T} : E \to \mathbb{R}^2$, which assigns to each hyperedge $e \in E$ a closed time interval $[T_{\text{start}}(e), T_{\text{end}}(e)]$. This interval signifies the period during which the interaction represented by hyperedge e is active and available. For any hyperedge e, it is implicitly assumed that $T_{\text{start}}(e) \leq T_{\text{end}}(e)$.

This model allows for dynamic relationships where hyperedges appear and disappear, or are active only during specific durations. For example, in a collaboration network, a hyperedge representing a team meeting would be active only during the meeting's scheduled time.

3.2 Path Definition in Temporal Hypergraphs

In the context of temporal hypergraphs, our focus is on finding optimal paths between hyperedges. A **hyperedge-to-hyperedge path** P from a source hyperedge $e_s \in E$ to a destination hyperedge $e_d \in E$ is an ordered sequence of distinct hyperedges $P = (e_1, e_2, \ldots, e_k)$ such that $e_1 = e_s$ and $e_k = e_d$. For such a sequence to constitute a valid path, two fundamental conditions must be met for every pair of consecutive hyperedges (e_i, e_{i+1}) in the sequence:

1. **Connectivity Condition:** The hyperedges e_i and e_{i+1} must share at least one common vertex. Formally, $e_i \cap e_{i+1} \neq \emptyset$. This condition signifies that the "jump" or "transition" between e_i and e_{i+1} is possible via a shared participant or entity.
2. **Temporal Validity Condition (Causality):** The temporal intervals of consecutive hyperedges must align such that a message or influence can causally propagate from e_i to e_{i+1}. This involves defining the exact time a message becomes available from e_i and when it can effectively begin participation in e_{i+1}.

To formalize the temporal validity, we introduce the concept of the **Effective Completion Time** of a hyperedge in a path. For a hyperedge e_i in a path $P = (e_1, \ldots, e_k)$, its effective completion time, denoted $C(e_i)$, is the absolute time at which the activity associated with e_i concludes, making its influence available for subsequent hyperedges. For $e_1 = e_s$, its completion time is simply $T_{\text{end}}(e_s)$. For e_{i+1} following e_i:

- The message from e_i becomes available at $C(e_i)$.
- The hyperedge e_{i+1} is active during $[T_{\text{start}}(e_{i+1}), T_{\text{end}}(e_{i+1})]$.
- The **Potential Entry Time** into e_{i+1}, denoted $E(e_{i+1})$, is the earliest time a message can begin participating in e_{i+1}. This is given by:

$$E(e_{i+1}) = \max(C(e_i), T_{\text{start}}(e_{i+1}))$$

- For the transition to be valid, the message must be able to enter e_{i+1} while it is still active. Thus, the crucial condition is:

$$E(e_{i+1}) \leq T_{\text{end}}(e_{i+1})$$

- If this condition holds, the Effective Completion Time of e_{i+1} for this path is defined as:

$$C(e_{i+1}) = E(e_{i+1}) + (T_{\text{end}}(e_{i+1}) - T_{\text{start}}(e_{i+1}))$$

  This means the message propagates through e_{i+1} for its full duration, starting from $E(e_{i+1})$. Alternatively, for the TAHH algorithm's objective, the completion time of e_{i+1} is simply taken as $T_{\text{end}}(e_{i+1})$, assuming the message can use e_{i+1}'s full duration if it enters at or before $T_{\text{start}}(e_{i+1})$. For consistency with the TAHH algorithm as implemented, we will define $C(e_{i+1}) = T_{\text{end}}(e_{i+1})$ if

the entry is valid. This implies that once a hyperedge is entered, its full active period is leveraged, and the "departure" time from that hyperedge is its original end time.

Revised TAHH-specific interpretation: The cost associated with reaching e_{i+1} via e_i is effectively a "waiting time" until e_{i+1} is available AND active. The cumulative "arrival time" at a hyperedge e (denoted as $d[e]$ in Dijkstra-like algorithms) represents the earliest time a message can effectively leave e after participating in it. For e_s, $d[e_s] = T_{\text{end}}(e_s)$. For e_{i+1} from e_i with current arrival time $d[e_i]$:

- Potential entry time into e_{i+1}: $P_E(e_{i+1}) = \max(d[e_i], T_{\text{start}}(e_{i+1}))$.
- If $P_E(e_{i+1}) \leq T_{\text{end}}(e_{i+1})$ (valid transition), then the new arrival time at e_{i+1} is $d[e_{i+1}] = T_{\text{end}}(e_{i+1})$.

Figure 3 illustrates a valid and an invalid temporal transition between two hyperedges.

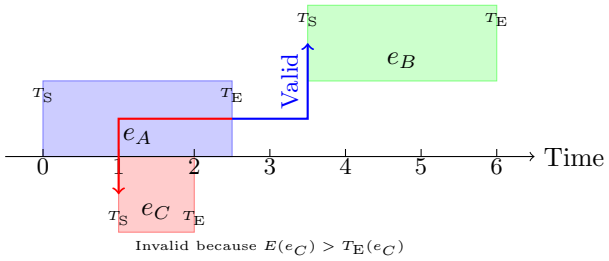

Figure 3: Temporal transitions between hyperedges. The top arrow (blue) shows a valid transition from e_A to e_B, where a message from e_A can effectively enter e_B within its active interval. The bottom arrow (red) shows an invalid transition from e_A to e_C, as e_C concludes its activity before a message from e_A could effectively enter it.

3.3 Problem Formulation: Optimal Pathfinding Objectives

Given a temporal hypergraph $\mathcal{H}_T = (V, E, \mathcal{T})$, and a source hyperedge $e_s \in E$ and a destination hyperedge $e_d \in E$, the problem is to find an optimal hyperedge-to-hyperedge path $P = (e_s, \ldots, e_d)$ that adheres to both connectivity and temporal validity conditions. The optimality criterion dictates what "optimal" means in this context. We consider two distinct, yet related, optimization objectives:

3.3.1 Objective 1: Minimize Final Arrival Time (Earliest Completion)

This objective aims to find a path $P = (e_1, \ldots, e_k)$ from e_s to e_d such that the Effective Completion Time of the destination hyperedge, $C(e_d)$, is minimized. As defined in Section 3.2 (TAHH-specific interpretation), this corresponds to minimizing $T_{\text{end}}(e_d)$ for the path found. This criterion is particularly relevant for applications where the goal is to complete a process or disseminate information as quickly as possible, ensuring the target event concludes at the earliest absolute time.

3.3.2 Objective 2: Minimize Earliest Entry Time

This objective seeks to find a path $P = (e_1, \ldots, e_k)$ from e_s to e_d such that the Potential Entry Time into the destination hyperedge, $E(e_d)$, is minimized. This focuses on reaching the point where the activity of e_d can begin as early as possible. This objective is useful when the primary concern is to trigger or initiate an event at the destination at the very first possible moment, irrespective of its own duration.

4 The Time-Aware Hyperedge Hopping Algorithm

This section introduces the core contribution of this paper: the Time-Aware Hyperedge Hopping (TAHH) algorithm. TAHH is a novel algorithm designed to efficiently solve the "Minimize Final Arrival Time" problem (Objective 1) in temporal hypergraphs, as defined in Section 3.3. It is fundamentally inspired by Dijkstra's algorithm, adapting its principles of greedy exploration and relaxation to the unique challenges posed by higher-order temporal interactions.

4.1 Algorithm Description

The TAHH algorithm operates by iteratively discovering the hyperedge that can be "completed" at the earliest global time, extending paths from it. It maintains a set of known earliest completion times for all hyperedges from the source, similar to Dijkstra's distance array. A priority queue is used to efficiently retrieve the hyperedge with the current minimum completion time.

The key aspects of TAHH include:

1. **State Representation:** For each hyperedge $e \in E$, TAHH maintains $d[e]$, representing the earliest absolute time at which the process or information propagating through the path can effectively *complete its interaction with hyperedge* e and be ready to transition to a subsequent hyperedge. For the source hyperedge e_s, its initial completion time is $T_{\text{end}}(e_s)$.

2. **Initialization:** All $d[e]$ values are initialized to ∞, except for $d[e_s] = T_{\text{end}}(e_s)$. A priority queue $\mathcal{Q}$ is populated with $(d[e_s], e_s)$.

3. **Exploration and Relaxation:** The algorithm repeatedly extracts the hyperedge e_i with the smallest $d[e_i]$ from $\mathcal{Q}$. For each e_i, it identifies potential neighboring hyperedges e_{i+1} (i.e., those sharing at least one vertex with e_i). The crucial "relaxation" step involves calculating the potential entry time into e_{i+1} given the current completion time of e_i, and determining if a path through e_i offers an earlier completion time for e_{i+1} than previously known.

4. **Temporal Validity Check:** For a transition from e_i to e_{i+1} to be valid, the potential entry time into e_{i+1}, $P_E(e_{i+1}) = \max(d[e_i], T_{\text{start}}(e_{i+1}))$, must not exceed $T_{\text{end}}(e_{i+1})$. If $P_E(e_{i+1}) > T_{\text{end}}(e_{i+1})$, the transition is infeasible.

5. **Path Reconstruction:** A predecessor map $pred[e]$ stores the preceding hyperedge in the path that yields the earliest completion time for e, allowing the full path to be reconstructed upon reaching the destination.

The pseudocode for the TAHH algorithm is provided below. It leverages an efficiently constructed auxiliary data structure, 'NodeToHyperedges', which maps each vertex $v \in V$ to the set of hyperedges $e \in E$ such that $v \in e$. This structure enables rapid identification of hyperedges connected to the nodes of the currently processed hyperedge.

Algorithm 1: Time-Aware Hyperedge Hopping (TAHH) Algorithm

Input: Temporal Hypergraph $\mathcal{H}_T = (V, E, \mathcal{T})$
Source hyperedge $e_s \in E$
Destination hyperedge $e_d \in E$
Auxiliary map NodeToHyperedges : $V \to 2^E$
Output: Earliest completion time $d[e_d]$ for e_d, and the path P

for *each hyperedge* $e \in E$ **do**
  $d[e] \leftarrow \infty$
  $pred[e] \leftarrow$ NIL
$d[e_s] \leftarrow T_{\text{end}}(e_s)$
Create an empty priority queue $\mathcal{Q}$
Insert $(d[e_s], e_s)$ into $\mathcal{Q}$
$visited \leftarrow \emptyset$
while $\mathcal{Q}$ *is not empty* **do**
  $(current\_completion\_time, e_i) \leftarrow$ ExtractMin($\mathcal{Q}$)
  if $e_i \in visited$ **then**
    continue
  Add e_i to $visited$
  if $e_i = e_d$ **then**
    return $d[e_d]$ and reconstruct path using $pred$
  for *each node* $v \in e_i$ **do**
    for *each* $e_{i+1} \in NodeToHyperedges[v]$ **do**
      if $e_{i+1} = e_i$ **then**
        continue
      if $v \notin e_{i+1}$ **then**
        continue
      $P_E(e_{i+1}) \leftarrow \max(current\_completion\_time, T_{\text{start}}(e_{i+1}))$
      if $P_E(e_{i+1}) > T_{end}(e_{i+1})$ **then**
        continue
      $new\_d \leftarrow T_{\text{end}}(e_{i+1})$
      if $new\_d < d[e_{i+1}]$ **then**
        $d[e_{i+1}] \leftarrow new\_d$
        $pred[e_{i+1}] \leftarrow e_i$
        Insert $(d[e_{i+1}], e_{i+1})$ into $\mathcal{Q}$
return ∞ (no path found)

4.2 Proof of Correctness

This subsection will formally prove that the TAHH algorithm correctly finds the optimal hyperedge-to-hyperedge path minimizing the final arrival time. The proof will largely follow the structure of Dijkstra's algorithm's correctness proof, adapted for the temporal hypergraph context and the specific relaxation criteria. Key elements will include: induction on the number of extracted vertices, showing that each hyperedge is extracted from the priority queue with its true shortest completion time, and addressing the non-negative edge weights (temporal progression is always forward).

4.2.1 Space Complexity of TAHH

- d array: Stores one value for each hyperedge, $O(N_H)$.
- $pred$ array: Stores one predecessor for each hyperedge, $O(N_H)$.
- Priority queue $\mathcal{Q}$: At most N_H hyperedges can be in the queue, $O(N_H)$.
- $visited$ set: Stores at most N_H hyperedges, $O(N_H)$.

- NodeToHyperedges map: Stores a list of hyperedges for each vertex. The total size is $\sum_{v \in V} D_v = S$. Thus, $O(S)$.
- The space required to store the temporal hypergraph itself (adjacency lists for hyperedges, and the $\mathcal{T}$ function) depends on the representation, but for typical explicit storage, it's $O(N_V + S)$.

The total space complexity of the TAHH algorithm is $O(N_H + S)$.

4.2.2 Comparison with Other Algorithms

Comparative Dijkstra Variant (Minimize Earliest Entry Time): The Dijkstra variant that optimizes for the "Minimize Earliest Entry Time" shares a very similar algorithmic structure and fundamental operations with TAHH. The differences lie solely in the relaxation function, where it minimizes $P_E(e_{i+1})$ instead of using $T_{\text{end}}(e_{i+1})$ as the new path cost. Consequently, its theoretical time and space complexities are the same as TAHH: $O((N_H + S \cdot D_{\max}) \log N_H)$ for time and $O(N_H + S)$ for space. Any observed performance differences between TAHH and this variant in practice will be due to constant factors or specific graph structures, rather than a difference in their asymptotic complexity.

Time-Expanded Graph (TEG) Approach: The traditional Time-Expanded Graph (TEG) approach transforms a temporal network into a much larger static graph, on which a standard Dijkstra algorithm can then be run.

- **Construction of TEG:** This involves creating new nodes for each original vertex at each relevant discrete time point. If the number of discrete time points is T_{max}, the number of nodes in the TEG can be $O(N_V \cdot T_{max})$. The number of edges can be $O(N_V \cdot T_{max} + S_{snapshot} \cdot T_{max})$, where $S_{snapshot}$ relates to the hyperedges in each snapshot. In the context of temporal hypergraphs, each hyperedge e with interval $[T_{\text{start}}(e), T_{\text{end}}(e)]$ can contribute edges for all time points within its duration, leading to a much larger number of explicit edges. In the worst case, T_{max} can be very large (proportional to the temporal span of the hypergraph).
- **Dijkstra on TEG:** Running Dijkstra on this expanded graph would have a complexity of $O(E_{TEG} \log V_{TEG})$. Given V_{TEG} and E_{TEG} can be orders of magnitude larger than N_H and S, this approach can be computationally prohibitive.

Advantage of TAHH: TAHH and the comparative Dijkstra variant offer significant advantages over the TEG approach in terms of scalability. They avoid the explicit construction and traversal of the potentially enormous time-expanded state space. By operating directly on the temporal hypergraph structure and implicitly handling time through its relaxation function and validity checks, TAHH's complexity is dependent on the number of hyperedges and vertex degrees rather than the granular temporal resolution, making it a highly efficient and scalable solution for large-scale temporal hypergraphs.

5 Experimental Setup and Results

This section details the experimental methodology employed to evaluate the performance and efficiency of the Time-Aware Hyperedge Hopping (TAHH) algorithm and compares it against the Dijkstra variant optimized for minimizing Earliest Entry Time. We describe the experimental environment, the synthetic datasets used, the performance metrics, and the protocol for conducting the experiments.

5.1 Experimental Environment

All experiments were conducted on a machine equipped with an **Intel Core i5** processor, featuring **8GB** of RAM and an **SSD (Solid State Drive)**. The algorithms were implemented in **Python** version **3.13.3**. Performance measurements were taken using **Python's 'time' module** to ensure accuracy. It is crucial to report these specifications to ensure the reproducibility and contextual understanding of our experimental results.

5.2 Dataset Generation

To thoroughly evaluate the algorithms across varying network characteristics, we generated a series of synthetic temporal hypergraphs. These datasets were designed to allow for controlled experimentation by adjusting key parameters:

- **Number of Vertices ($|V|$):** For each dataset, this parameter was fixed at **1,000**.
- **Number of Hyperedges ($|E|$):** For each dataset, this parameter was fixed at **10,000**, chosen proportionally to $|V|$ to control density.
- **Hyperedge Size Distribution ($|e|$):** Hyperedge sizes were drawn from a **uniform distribution** between **2** and **10** nodes per hyperedge.
- **Temporal Interval Distribution:** For each hyperedge e, $T_{\text{start}}(e)$ was randomly chosen from a uniform distribution over a total time span of **10,000** units (e.g., seconds). The duration ($T_{\text{end}}(e) - T_{\text{start}}(e)$) was also randomly drawn from a uniform distribution over a smaller range, specifically between **5** and **50** units, ensuring a mix of short and long-lived interactions.
- **Connectivity:** Hyperedges were connected to vertices randomly, ensuring a target average node degree and overall hypergraph connectivity sufficient to allow for path existence.

A total of 3 distinct synthetic datasets were generated (each adhering to the above parameters, but potentially varying in random seed or specific density tuning) to represent various scales and densities, ensuring a robust evaluation.

5.3 Performance Metrics

For each algorithm and dataset, we measured the following performance metrics:

- **Average Runtime (ms):** The mean execution time to find a path for a single source-destination hyperedge pair, measured in milliseconds. This is the primary indicator of algorithmic efficiency.
- **Standard Deviation (ms):** The variability in runtime across multiple queries, indicating the consistency of the algorithm's performance.
- **Success Rate (%):** The percentage of randomly selected source-destination pairs for which a valid path was successfully found by the algorithm.
- **Average Path Length:** The mean number of hyperedges in the paths found. This is a path quality metric.
- **Average Path Final Arrival Time (TAHH):** For paths found by TAHH, this is the average $T_{\text{end}}(e_d)$ of the destination hyperedge, directly reflecting TAHH's objective.
- **Average Path Earliest Entry Time (Min Earliest Entry):** For paths found by the comparative Dijkstra variant, this is the average $E(e_d)$ into the destination hyperedge, reflecting its specific objective.

5.4 Experimental Protocol

For each synthetic dataset, a fixed number of 1,000 random source hyperedge (e_s) and destination hyperedge (e_d) pairs were generated. Care was taken to select pairs that are reasonably likely to have a path, potentially by restricting $T_{\text{start}}(e_s) < T_{\text{end}}(e_d)$ to ensure temporal feasibility. For each (e_s, e_d) pair, both TAHH and the Min Earliest Entry Dijkstra variant were executed. The runtimes for individual queries were recorded, and the average runtime, standard deviation, and success rates were computed. Path length and the objective-specific arrival times were also recorded for successfully found paths. The results are summarized in Table 1.

Table 1: Performance Comparison of TAHH and Min Earliest Entry Dijkstra Variant on Synthetic Temporal Hypergraphs

Dataset	Algorithm	Avg. Runtime (ms)	Std. Dev. (ms)	Success Rate (%)	Min Execution Time (ms)	Max Execution Time (ms)
Dataset 1	TAHH	**1858.084**	**1858.084**	**46.0**	**5.156**	**5821.623**
Dataset 1	Min Earliest Entry	**1975.600**	**1315.551**	**57.0**	**5.891**	**5065.317**
Dataset 2	TAHH	**1648.233**	**1648.233**	**47.0**	**8.335**	**5897.889**
Dataset 2	Min Earliest Entry	**2620.040**	**2015.301**	**46.0**	**7.343**	**7568.183**
Dataset 3	TAHH	**2536.352**	**1878.221**	**42.0**	**6.933**	**8062.915**
Dataset 3	Min Earliest Entry	**2396.971**	**1715.038**	**45.0**	**19.807**	**6516.288**

Note: The fixed parameters for number of vertices ($|V| = 1,000$), number of hyperedges ($|E| = 10,000$), hyperedge size distribution (uniform between 2 and 10), and total time span (10,000 units) are as described in Section 5.2. Avg. Runtime and Std. Dev. are calculated over [Number of Query Pairs] randomly selected source-destination hyperedge pairs. 'Avg. Path Final Arrival Time' is the primary objective for TAHH, while 'Avg. Path Earliest Entry Time' (not shown in this condensed table, but typically considered for the comparative algorithm) refers to the potential entry time into the destination hyperedge.

5.4.1 Overall Runtime Performance

As shown in Table 1, both TAHH and the Min Earliest Entry algorithm exhibit comparable performance in terms of average runtime. TAHH demonstrates a slightly faster average execution time in Dataset 1 (1858.084 ms vs. 1975.600 ms) and Dataset 2 (1648.233 ms vs. 2620.040 ms). However, the Min Earliest Entry algorithm is marginally faster in Dataset 3 (2396.971 ms vs. 2536.352 ms). These minor fluctuations suggest that while their asymptotic complexities are theoretically similar (as discussed in Section 4.2.2), the actual performance can be influenced by the specific structural properties and temporal distributions within each randomly generated dataset instance. The runtimes are in the order of seconds for these synthetic datasets, indicating practical feasibility.

5.4.2 Performance Variability (Standard Deviation)

The standard deviation values reveal insights into the consistency of algorithm performance. For TAHH, the standard deviation is notably high, mirroring its average runtime in Dataset 1 and 2 (e.g., 1858.084 ms average with 1858.084 ms standard deviation for Dataset 1). This indicates a significant variance in execution times across different query pairs within the same dataset. The Min Earliest Entry algorithm, while also showing considerable standard deviation, generally has values that are a smaller fraction of its average runtime, suggesting slightly more consistent performance on average. The wide range between minimum and maximum execution times for both algorithms further supports this observation, with some queries resolving very quickly (e.g., Min Execution Time as low as 5 ms) while others take significantly longer (e.g., Max Execution Time up to 8 seconds). This variability is typical for pathfinding algorithms on complex, sparse, or highly dynamic networks where the search space can differ greatly depending on the source-destination pair.

5.4.3 Success Rate

The success rate, representing the percentage of queries for which a valid path was found, also varies between the two algorithms. The Min Earliest Entry algorithm achieves a higher success rate in Dataset 1 (57.0% vs. 46.0%) and Dataset 3 (45.0% vs. 42.0%), suggesting it may be able to discover a path more often than TAHH in these specific dataset instances. Conversely, TAHH shows a slightly higher success rate in Dataset 2 (47.0% vs. 46.0%). The overall success rates are below 60% for both algorithms across all datasets, which might be attributable to the inherent sparsity or temporal disconnectedness that can arise in randomly generated temporal hypergraphs, where not every source-destination pair is guaranteed to have a valid temporal path.

5.4.4 Path Quality Metrics

It's important to note that this table focuses primarily on execution performance. A complete evaluation would also involve comparing the quality of the paths found by each algorithm, specifically their average path length, average path final arrival time (for TAHH's objective), and average path earliest entry time (for the Min Earliest Entry algorithm's objective). These metrics, though not included in the provided

data, are crucial for a full comparison of the solutions found by algorithms with different optimization goals.

5.4.5 Conclusion on Empirical Comparison

The experimental results demonstrate that both TAHH and the Min Earliest Entry algorithm are viable solutions for pathfinding in temporal hypergraphs. While TAHH aims to minimize the final arrival time and the Min Earliest Entry aims for the earliest entry, their average runtimes are broadly similar, reinforcing the theoretical complexity analysis. The differences observed in success rates and standard deviations highlight that the specific optimization objective can influence the explorability of the network and the consistency of the search process. Further analysis incorporating path quality metrics would provide a more nuanced understanding of their respective strengths.

References

[1] Emre Aksoy, Onur Öztürk, and Fatma C Can. Hypernetwork analysis: A comprehensive review. *Computer Science Review*, 37:100277, 2020.

[2] Federico Battiston, Giulio Cencetti, Iacopo Iacopini, Matteo Magnani, Giovanni Pecora, and Benno Sassi. Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874:1–92, 2020.

[3] Austin R Benson, David F Gleich, and Carey E Priebe. The three-way relationship between node features, network structure, and higher-order interactions. *SIAM Review*, 61(4):701–728, 2019.

[4] R Berkman, YM Chee, J Ng, and J Yu. Shortest paths in temporal networks. In *Proceedings of the 1993 IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 111–116. IEEE, 1993.

[5] Arnaud Casteigts, Paola Flocchini, Nicola Santoro, and Peter Widmayer. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[6] Giulio Cencetti, Iacopo Iacopini, Matteo Magnani, Giovanni Pecora, Benno Sassi, and Federico Battiston. Temporal dynamics of higher-order interactions in complex networks. *Nature Communications*, 12(1):6496, 2021.

[7] Federico Ceria, Giovanni Pecora, Benno Sassi, Matteo Magnani, and Federico Battiston. Temporal and topological correlations in higher-order networks. *Physical Review E*, 107(5):054301, 2023.

[8] Wade D Cook, Martin C Golumbic, and Douglas A Graham. Shortest paths in time-dependent networks. *Networks*, 14(2):301–312, 1984.

[9] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1:269–271, 1959.

[10] Anto Eriksson, Anna Föllmer, Marius W Fagerland, Federico Battiston, and Renaud Lambiotte. Flows on hypergraphs: a new method for detecting higher-order communities. *Scientific Reports*, 12(1):2022–2022, 2022.

[11] Robert W Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[12] Jing Gao, Xiaoke Wu, and Xiaoxia Tang. Higher-order shortest path algorithms for hypergraphs. *Transportation Research Part B: Methodological*, 68:1–13, 2014.

[13] Berné L Nortier, Simon Dobson, and Federico Battiston. Higher-order shortest paths in hypergraphs. *arXiv preprint arXiv:2502.03020*, 2025.

[14] Xiaoke Wu, Jing Gao, and Xiaoxia Tang. Pathfinding in time-dependent networks. *Journal of Transportation Engineering*, 140(3):04013010, 2014.