**TEMPORAL REASONING ALGORITHMS**

**JESSICA FLEISCHER, FRANCESCA LUCCHETTI, MUHTASIM MIRAZ, BENJAMIN PRUD’HOMME, LUKE HUNSBERGER**

### INTRODUCTION

Temporal networks are data structures for representing and reasoning about temporal constraints on real events and activities. The most basic kind of temporal network is a Simple Temporal Network (STN) which can accommodate such constraints as release times, deadlines, precedence constraints, and duration constraints. The primary goal of this project was to create a public repository that will include implementations of several useful algorithms for manipulating STNs. Researchers interested in using STNs as a temporal reasoning tool will be able to use this repository for their needs.

A Simple Temporal Network (STN) has two main components: a set $\mathcal{T}$ of real-valued variables called time-points, and a set $\mathcal{C}$ of edges imposing constraints on the distance between pairs of time-points. These constraints have the form $Y - X \leq \delta$, where $X$ and $Y$ are time-points, and $\delta$ is a real number.

The Simple Temporal Problem is that of determining if an STN is consistent - that is, whether there is a set of values for all time-points that satisfies all edge constraints. A consistent STN has an $n \times n$ matrix $D$ called the distance matrix.

For each pair of time-points $X$ and $Y$, the entry $D(X,Y)$ contains the length of the shortest path from $X$ to $Y$ in the STN graph.

### RESULTS

The consistency-checking algorithms for solving the Simple Temporal Problem are displayed in graphs (d) and (e). On both very sparse and dense networks, a “stop early” version of Bellman-Ford is our fastest algorithm, followed closely by Yen and Randomized Yen, which have similar performances due to the randomized generation of the networks. A surprising result is that the Bellman-Ford competitor based on the AddToFeasible algorithm performs better than Bellman-Ford itself on dense networks of around or over 400 time-points. Another surprising result was the performance of Directional Path Consistency (DPC), which surpassed even Floyd-Warshall’s complexity of $O(n^3)$. We concluded that this was a consequence of the new edges inserted by the DPC algorithm into the network. As we cannot predict how many edges DPC will add into a random network, the algorithm may perform better or worse than expected.

Graph (f) shows a comparison of the performances of Floyd-Warshall’s and Johnson’s algorithms on very sparse and dense networks. Both of these algorithms generate distance matrices, with the difference that Johnson works by combining two other algorithms, Dijkstra and Bellman-Ford. We calculate the complexity of Johnson’s algorithm from the $n$ iterations of Dijkstra within its body: $O(n(\log n + mn))$. In sparse graphs where the number of edges $m$ is much less than the maximum value $n^2$, Johnson’s algorithm should run faster than Floyd-Warshall’s. This is confirmed by our results, which show that Johnson performs worse than Floyd-Warshall on denser graphs.

### CITATIONS