University of St Andrews | FOUNDED 1413

Exploring temporal graphs with frequent edges

Duncan Adamson

20 March 2025

Temporal Graphs

- A **temporal graph** is a generalisation of (static) graphs when, rather than one single set of edges, there is an **ordered sequence** of edge sets.
- By convention:

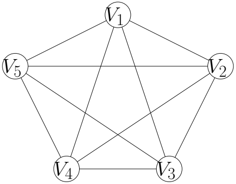
$$\mathcal{G} = (V, E_1, E_2, \dots, E_T)$$

where:
  - V is the set of vertices,
  - $E_i \subseteq V \times V$ is a set of edges.
- We call each set of edges a **time step**, and the number of sets as the **lifetime** of the graph.
- An edge e is **active** in timestep i iff $e \in E_i$.
- The **underlying graph** is the static graph

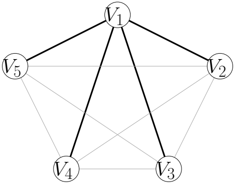
$$U(\mathcal{G}) = \left(V, \bigcup_{i \in [1,T]} E_i \right).$$

Temporal Graph Example

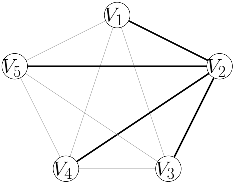
$$\mathcal{G} = (\{v_1, v_2, v_3, v_4, v_5\}, E_1, E_2, E_3, E_4, E_5)$$



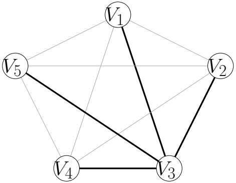
Underlying Graph



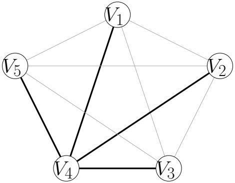
$E_1 = (v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5)$



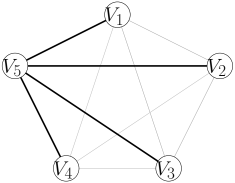
$E_2 = (v_1, v_2), (v_2, v_3), (v_2, v_4), (v_2, v_5)$



$E_3 = (v_1, v_3), (v_2, v_3), (v_3, v_4), (v_3, v_5)$



$E_4 = (v_1, v_4), (v_2, v_4), (v_3, v_4), (v_4, v_5)$



$E_5 = (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5)$

Temporal Graphs with Regular Edges

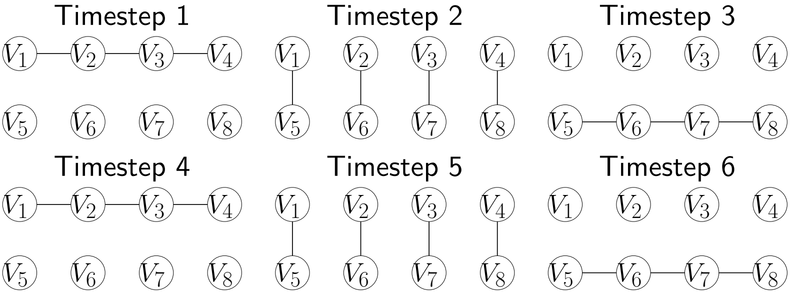
Definition

An edge e in a temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ is *r-regular* if $e \in E_t$ iff $e \in E_{t+r \bmod T}$, $\forall t \in [1, T]$.
The *regularity* of an edge e, denoted r_e, is the smallest value for which e is r_e-regular.

Definition

A temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ is *r-regular* if every edge $e \in \bigcup_{t \in [T]}$ is r-regular.

Temporal Graphs with Regular Edges



Example of a 3-regular temporal graph.

Temporal Graphs with Frequent Edges

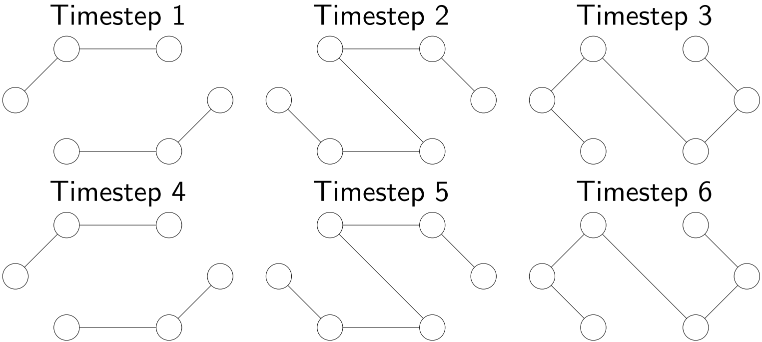
Definition

An e in a temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ is *f-frequent* iff $e \in \bigcup_{\tau \in [t, t+f]} E_\tau$, $\forall t \in [1, T - f]$.
The *frequency* of an edge e, denoted f_e, is the smallest value such that e is f_e-frequent.

Definition

A temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ is *f-frequent* if every edge $e \in \bigcup_{t \in [1, T]} E_t$ is f-frequent.

Temporal Graphs with Frequent Edges



Example of a 2-frequent and 3-regular temporal graph.

Frequency and Regularity

Observation

Every r-regular temporal graph is an r-frequent temporal graph.

Observation

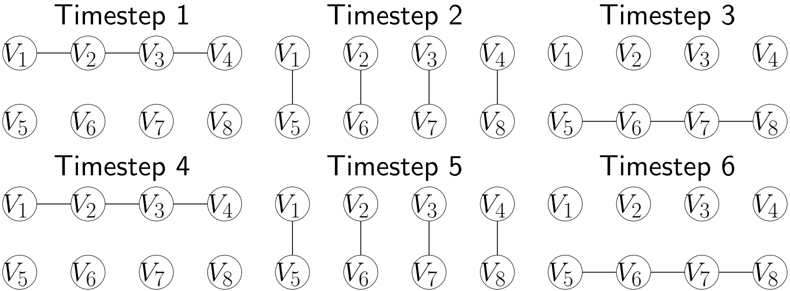
Any upper bound on the exploration of f-frequent temporal graphs is also an upper bound on the exploration of f-regular temporal graphs.

Any lower bound on the exploration of r-regular temporal graphs is also a lower bound on the exploration of r-frequent temporal graphs.

Temporal Walks

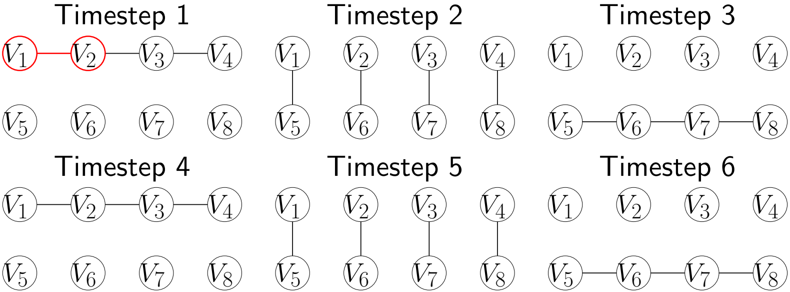
- A **temporal walk** is the temporal analogue of a walk in a static graph.
- A temporal walk on the temporal graph $\mathcal{G}$ is an ordered sequence of edges, $e_1, e_2, \ldots, e_k$, and timesteps $t_1, t_2, \ldots, t_k$ such that:
  - $e_1, e_2, \ldots, e_k$ form a walk in $U(\mathcal{G})$, and,
  - e_i is active in timestep t_i,
  - $1 \leq t_1 < t_2 < \cdots < t_k \leq T$
- Given an agent following a temporal walk, if there is some timestep t that does not appear in the sequence of timesteps, we say the agent is **waiting**.

Temporal Walks Example



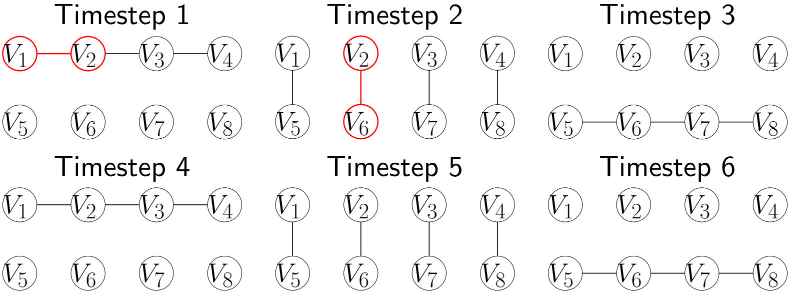
Temporal Walks Example

$((v_1, v_2), 1)$



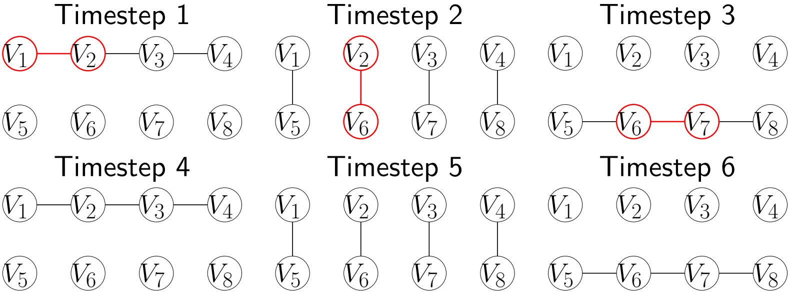
Temporal Walks Example

$((v_1, v_2), 1), ((v_2, v_6), 2)$



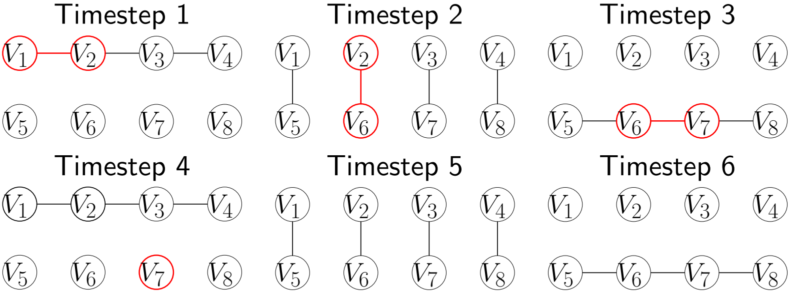
Temporal Walks Example

$((v_1, v_2), 1), ((v_2, v_6), 2), ((v_6, v_7), 3)$



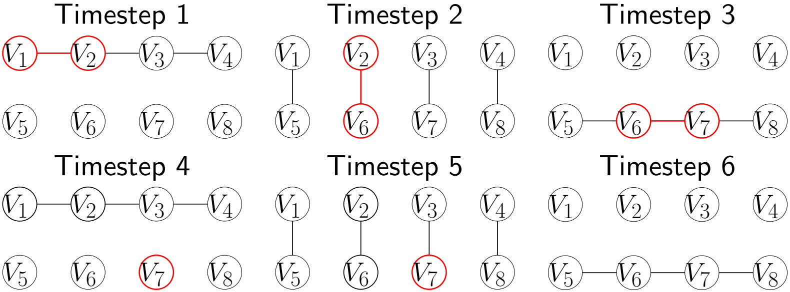
Temporal Walks Example

$((v_1, v_2), 1), ((v_2, v_6), 2), ((v_6, v_7), 3)$



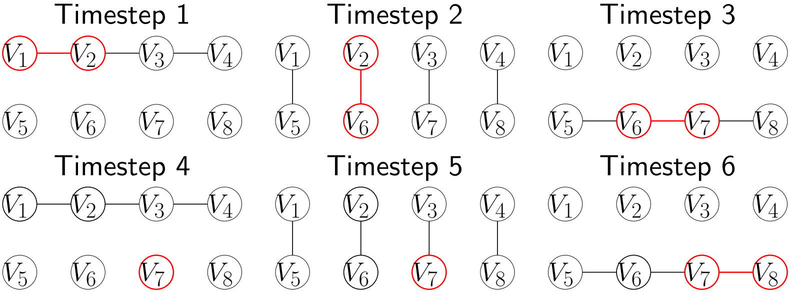
Temporal Walks Example

$((v_1, v_2), 1), ((v_2, v_6), 2), ((v_6, v_7), 3)$



Temporal Walks Example

$((v_1, v_2), 1), ((v_2, v_6), 2), ((v_6, v_7), 3), ((v_7, v_8), 6)$



Problems on Temporal Graphs

- Most algorithmic work on temporal graphs focuses on questions of **reachability** and **exploration**.
- **Reachability:** Given a temporal graph $\mathcal{G}$ and pair of vertices v_i, v_j, does there exist a temporal walk from v_i to v_j?
  - Optimisation Variant: What is the earliest an agent starting at v_i can reach v_j?
- **Exploration:** Given a temporal graph $\mathcal{G}$, does there exist a temporal walk visiting every vertex at least once?
  - Optimisation Variant: What is the earliest an agent starting at v_i can visit every vertex in the graph?

Results

Main Claim

Theorem

Any f-frequent temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ can be explored in at most $f(2|V| - 3)$ timesteps.

Basic Algorithm

- We find a *spanning tree*, $T = (V, E')$ on the underlying graph $U(\mathcal{G})$.
- Any walk $W = e_1, e_2, \dots, e_m$ exploring T will also explore $U(\mathcal{G})$, thus, by converting it to a temporal walk $\mathcal{W}$, we get a walk exploring $\mathcal{G}$.
- We convert as follows:
    - Let t_1 be the first timestep in which e_1 is active, i.e. th value such that $e_1 \in E_{t_1}$ and $\forall t \in [1, t_1 - 1]$, $e_1 \notin E_t$.
        - Note, $t_1 \leq f$.
    - In general, let $t_i \in [t_{i-1} + 1, t_{i-1} + f]$ be the value such that $e_i \in E_{t_i}$ and, $\forall t \in [t_{i-1} + 1, t_i - 1]$, $e_t \notin E_t$.
    - As there are at most $2|V| - 3$ edges in W, and $t_i \leq t_{i-1} + f$, the total number of timesteps needed to complete the walk is $f(2|V| - 3)$.

Stronger and More General

- Theorem 5 gives a good general outline, but assumes the worst, i.e. that every edge has a frequency of exactly f.
- In general, we want to look at graphs where different edges have different frequencies.
- To this end, we introduce the *frequency-weighted graph*, $\mathcal{F}(\mathcal{G})$, defined by the weighting function Weight : $E \mapsto \mathbb{N}$ on the underlying graph $U(\mathcal{G}) = (V, E_1, E_2, \ldots, E_T)$, where $\text{Weight}(e) = f_e$, for any edge $e \in E$.

Exploring in the general setting

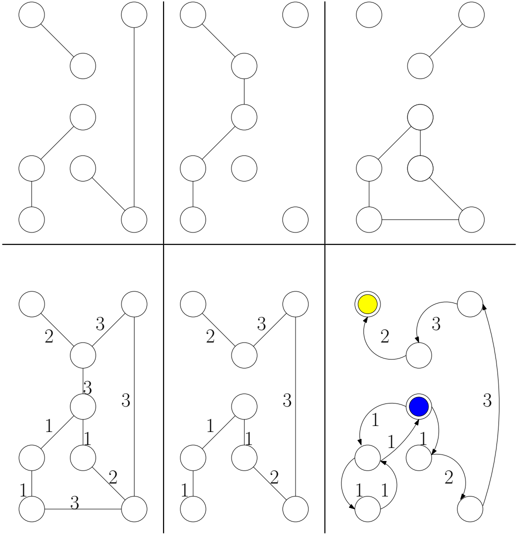
Theorem

Given a temporal graph $\mathcal{G}$, $\mathcal{G}$ can be explored in $2F$ timesteps, where F is the weight of the minimum weight spanning tree on $\mathcal{F}(\mathcal{G})$.

Outline

- Theorem 6 follows the same pattern as Theorem 5. We find a spanning tree, construct a walk exploring that tree, and convert this walk into a temporal walk exploring $\mathcal{G}$.
- The difference is that, rather than finding any spanning tree, we now want a *minimum-weight* spanning tree, with the weight defined by the frequency of the edges.
  - In essence, we want a tree with a lot of frequent edges if possible.
- As in the first case, we are waiting at most $f_e - 1$ timesteps for the edge e to activate (along with an additional timestep to transition across the edge).
- Therefore, the total time to complete the exploration is (roughly) $2F$, giving the theorem.

Example



Corollaries

Corollary

Any r-regular temporal graph $\mathcal{G} = (V, E_1, E_2, \dots, E_T)$ can be explored in at most $r(2|V| - 3)$ timesteps.

Corollary

Given a temporal graph $\mathcal{G}$, $\mathcal{G}$ can be explored in $2R$ timesteps, where R is the weight of the minimum weight spanning tree on $\mathcal{R}(\mathcal{G})$[1].

[1] The regularity based analogue of $\mathcal{F}(\mathcal{G})$

Motivating Examples

Sequential Connection Graphs

- A temporal graph is a *sequential connection graph* if there exists, for each vertex v, a permutation of all (outgoing) edges from v, $e_1, e_2, \ldots, e_{d(v)}$ such that the i^{th} e_i is active at timestep t iff $t \bmod d(v) \equiv i$.
  - This can be generalised to allow for multiple edges assigned to each vertex to be active in any given timestep.
- These graphs can be used to model systems where only one port at a time is checked, with some period between messages allowed for processing.
- **Key observation**: Each edge must, therefore, have frequency (at most) $d(v)$, where $d(v)$ denotes the degree of the vertex v.

Sequential Connection Graphs

Theorem

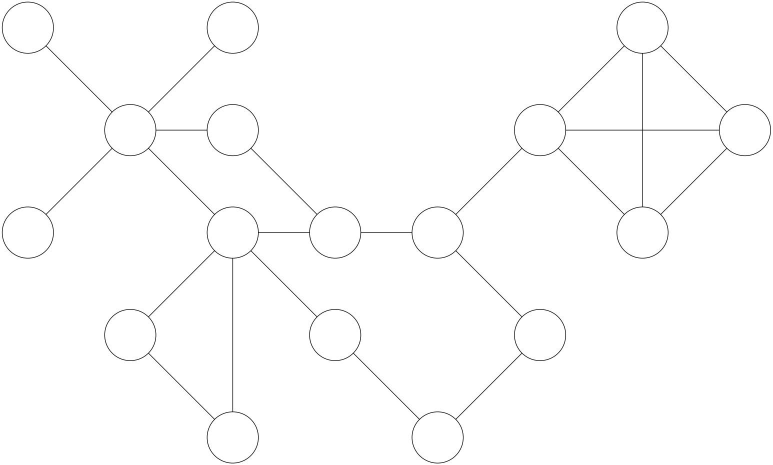
Given a sequential connection graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$, $\mathcal{G}$ can be explored in at most $4|E|$ timesteps, where $E = \bigcup_{t \in [T]} E_t$.

- From before, each edge has a frequency of at most $d(v)$.
- Therefore, the weight of the spanning tree will be at most $2|E|$.
- Hence, by Theorem 6, we get the bound.

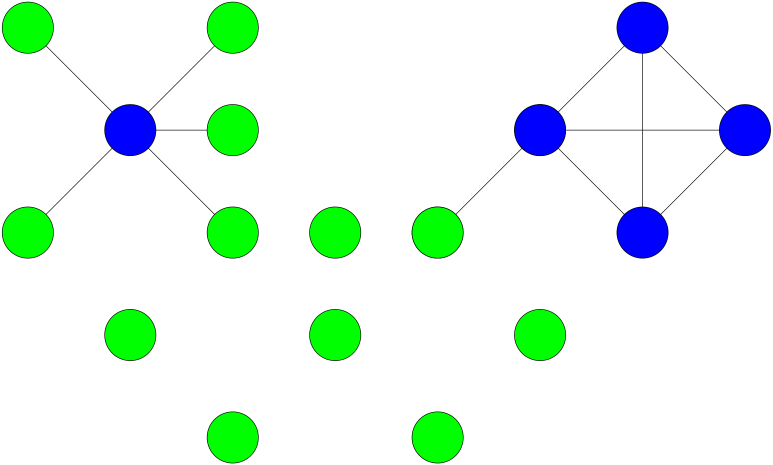
Broadcast Networks

- Based on ideas from distributed computing, a broadcast network graph is a symmetric directed temporal graph in which, at each timestep, either every outgoing edge from any given vertex is active, or none are.
  - This corresponds to the node either sending a message to all neighbours, or to none.
- We add the additional restriction that no node can activate (send messages) until *every* neighbour has sent one.
  - This models some sense of synchronisation in the network. A node does not necessarily care about what is going on far away, but it does need to ensure that the neighbours have received the previous message.
- Finally, we assume at least one node is broadcasting at each timestep.

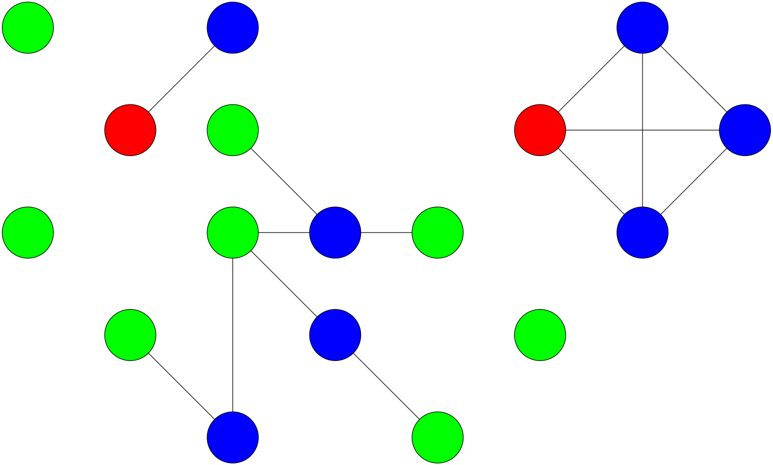
Broadcast Networks



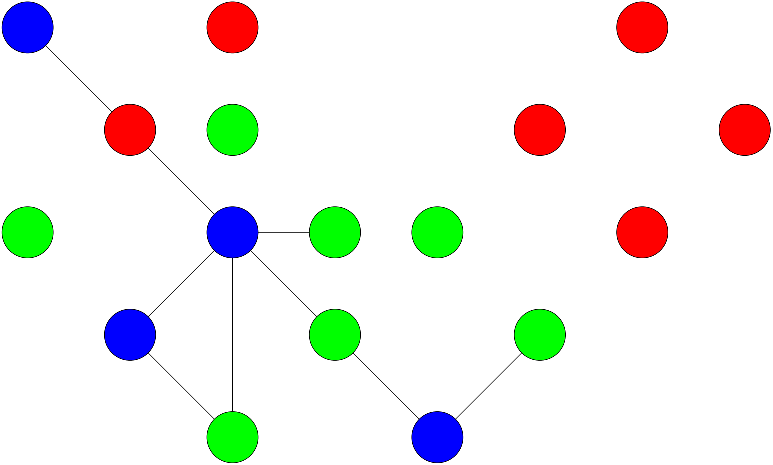
Broadcast Networks



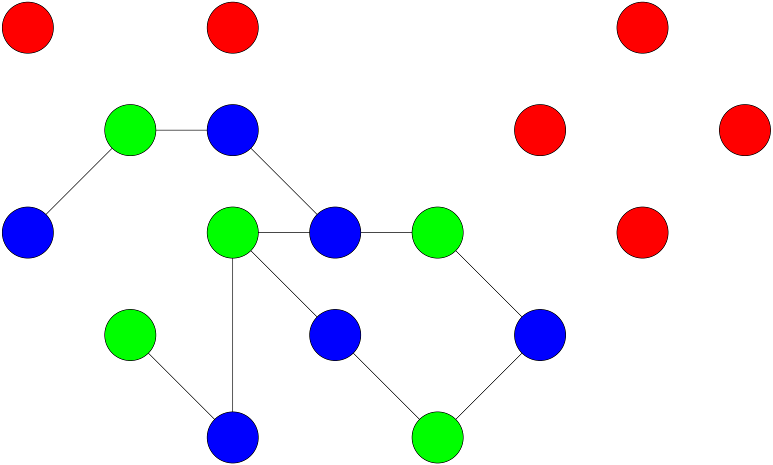
Broadcast Networks



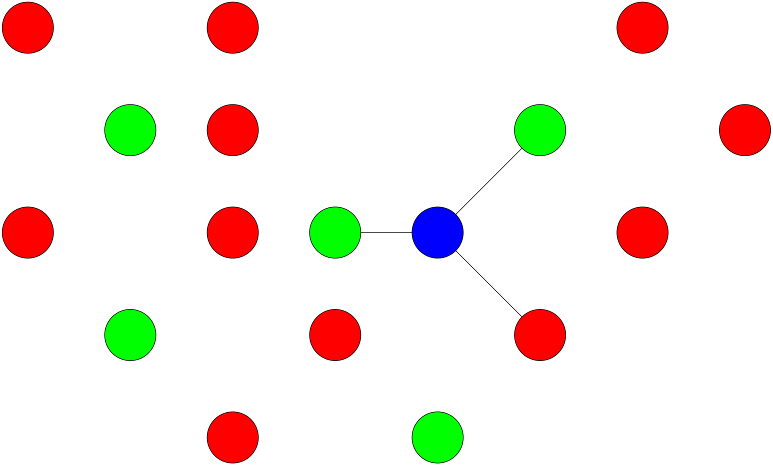
Broadcast Networks



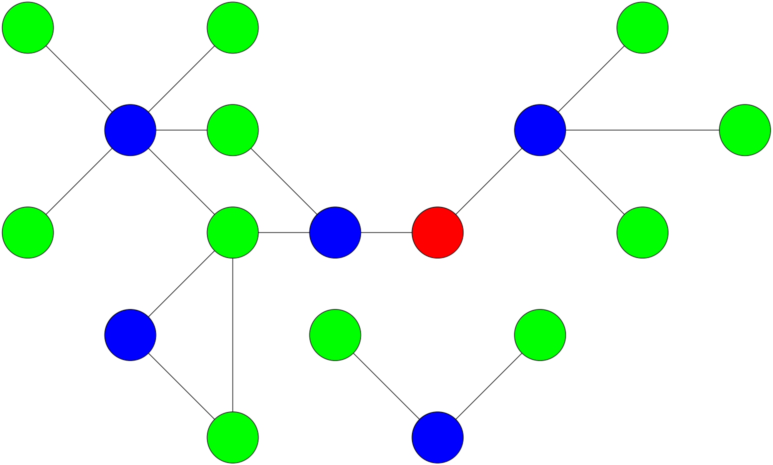
Broadcast Networks



Broadcast Networks



Broadcast Networks



Results for Broadcast Networks

Theorem

Any broadcast network temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ such that each timestep forms a connected graph can be explored in $\delta(2|V| - 3)$ timesteps, where δ is the smallest degree of any vertex in $U(\mathcal{G})$.

Outline:

- To be connected at each timestep, each vertex needs to either be active, or have some neighbour active.
- Since the vertex v has to wait for each neighbour to activate, and must be active iff no neighbour is, v has a frequency of at most $d(v)$.
- But, v also has to have a frequency of at most the frequency of all neighbours, so $f_v \leq d(u), \forall u \in N(v)$.
- Extrapolating across the graph gives the claim.

Results for Broadcast Networks

Theorem

Any broadcast network temporal graph $\mathcal{G} = (V, E_1, E_2, \ldots, E_T)$ such that each timestep forms a connected graph can be explored in $d|V|(2|V| - 3)$ timesteps, where d is the diameter of $U(\mathcal{G})$.

Outline:

- The goal is to force some node v to be able to activate only once every $d \cdot n$ timesteps.
- Starting with some node u at a distance of d, we activate u at timestep 1.
- Then, in timesteps $2, 3, \ldots, d(u)$, we activate one of u's neighbours, followed by activating u again.
- Repeating this, we activate the nodes at increasing distances from u, then back track to u, meaning we have to wait at most dn timesteps until we are forced to activate v.

Open Problems

- Does there exist a better algorithm for exploring graphs with frequent edges?
    - The worst case will be the same, but we may be able to formalise a stronger claim as an approximation of the optimal result.
- Can we tighten the bound on the frequency of (general) Broadcast Networks?

Open Problems

- Does there exist a better algorithm for exploring graphs with frequent edges?
  - The worst case will be the same, but we may be able to formalise a stronger claim as an approximation of the optimal result.
- Can we tighten the bound on the frequency of (general) Broadcast Networks?
- Thanks for listening!
- If you are interested, I am always looking for new collaborators.