Scalable Distributed Topologies

Carlos Baquero
DEI, FEUP, Universidade do Porto

MEIC SDLE 2021

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

- Graphs can be directed or undirected (in which case edges are bi-directional).

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

- Graphs can be directed or undirected (in which case edges are bi-directional).
- A Simple Graph is an undirected graph with no loops and no more than one edge between any two different vertices.

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

- Graphs can be directed or undirected (in which case edges are bi-directional).
- A Simple Graph is an undirected graph with no loops and no more than one edge between any two different vertices.
- Having an edge (x, y), we say that those vertices are adjacent (or neighbours).

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

- Graphs can be directed or undirected (in which case edges are bi-directional).
- A Simple Graph is an undirected graph with no loops and no more than one edge between any two different vertices.
- Having an edge (x, y), we say that those vertices are adjacent (or neighbours).
- A Weighted Graph is obtained by assigning a weight to each edge.

Graphs

A graph $G(V, E)$ can be defined by a set of vertices, and a set of edges that connect pairs of vertices. For instance, a path $G(V = \{a, b, c, d\}, E = \{(a, b), (b, c), (c, d)\})$ can be made into a ring by adding one edge connecting its ends $E = E \cup \{(d, a)\}$.

- Graphs can be directed or undirected (in which case edges are bi-directional).
- A Simple Graph is an undirected graph with no loops and no more than one edge between any two different vertices.
- Having an edge (x, y), we say that those vertices are adjacent (or neighbours).
- A Weighted Graph is obtained by assigning a weight to each edge.
- Path is a sequence of vertices, $\ldots, v_i, v_{i+1}, \ldots$, with edges connecting them $(v_i, v_{i+1}) \in E$.

Carlos Baquero
DEI, FEUP, Universidade do Porto

Walk Edges and vertices can be repeated

Trail Only vertices can be repeated

Path No repeated vertices or edges

Graphs

More topologies

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.

Graphs

More topologies

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.
- Connected Graph. There is a path between any two nodes.

Graphs

More topologies

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.
- Connected Graph. There is a path between any two nodes.
- Star. A "central" vertice and many leaf nodes connected to center

Graphs

More topologies

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.
- Connected Graph. There is a path between any two nodes.
- Star. A "central" vertice and many leaf nodes connected to center
- Tree. A connected graph with no cycles.

Graphs

More topologies

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.
- Connected Graph. There is a path between any two nodes.
- Star. A "central" vertice and many leaf nodes connected to center
- Tree. A connected graph with no cycles.
- Planar Graph. Vertices and edges can be drawn in a plane and no two edges intersect. E.g. Rings and Trees are planar.

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Complete Graph. Each pair of vertices has an edge connecting them. In a sub-graph we might find a clique with that property.
- Connected Graph. There is a path between any two nodes.
- Star. A "central" vertice and many leaf nodes connected to center
- Tree. A connected graph with no cycles.
- Planar Graph. Vertices and edges can be drawn in a plane and no two edges intersect. E.g. Rings and Trees are planar.

In a network context, graphs with cycles allow multi-path routing. This can be more robust but data handling can become more complex. Thus, distributed algorithms often construct trees to avoid cycles, while others try to work under multi-path.

Graphs
More concepts

Carlos Baquero
DEI, FEUP,
Universidade do Porto

- Connected Component. Maximal connected subgraph of G.

Graphs
More concepts

Carlos Baquero
DEI, FEUP,
Universidade do Porto

- Connected Component. Maximal connected subgraph of G.
- Degree of v_i. Number of adjacent vertices to v_i. In directed graphs there is in-degree and out-degree.

Graphs

More concepts

- Connected Component. Maximal connected subgraph of G.
- Degree of v_i. Number of adjacent vertices to v_i. In directed graphs there is in-degree and out-degree.
- Distance $d(v_i, v_j)$. Length of the shortest path connecting those nodes.

- Connected Component. Maximal connected subgraph of G.
- Degree of v_i. Number of adjacent vertices to v_i. In directed graphs there is in-degree and out-degree.
- Distance $d(v_i, v_j)$. Length of the shortest path connecting those nodes.
- Eccentricity of v_i. $ecc(v_i) = max(\{d(v_i, v_j) | v_j \in V\})$.
- Diameter. $D = max(\{ecc(v_i) | v_i \in V\})$
- Radius. $R = min(\{ecc(v_i) | v_i \in V\})$

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Connected Component. Maximal connected subgraph of G.
- Degree of v_i. Number of adjacent vertices to v_i. In directed graphs there is in-degree and out-degree.
- Distance $d(v_i, v_j)$. Length of the shortest path connecting those nodes.
- Eccentricity of v_i. $ecc(v_i) = max(\{d(v_i, v_j)|v_j \in V\})$.
- Diameter. $D = max(\{ecc(v_i)|v_i \in V\})$
- Radius. $R = min(\{ecc(v_i)|v_i \in V\})$
- Center. $\{v_i|ecc(v_i) == R\}$
- Periphery. $\{v_i|ecc(v_i) == D\}$

Carlos Baquero
DEI, FEUP, Universidade do Porto

- How big is the center of a tree?

Carlos Baquero
DEI, FEUP, Universidade do Porto

Graphs
Questions

- How big is the center of a tree?
- How big is the center of a path?

- How big is the center of a tree?
- How big is the center of a path?
- How big is the periphery of a ring? and the center?

Carlos Baquero
DEI, FEUP, Universidade do Porto

- How big is the center of a tree?
- How big is the center of a path?
- How big is the periphery of a ring? and the center?
- How to compute eccentricities?

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Random geometric. Vertices are dropped randomly uniformly into a unit square and adding edges to connect any two points within a given euclidean distance.

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Random geometric. Vertices are dropped randomly uniformly into a unit square and adding edges to connect any two points within a given euclidean distance. Is it planar?
- Random Erdos-Renyi. $G(n, p)$ model, n nodes are connected randomly. Each edge is included with independent probability p.

Graphs

More complex topologies

- Random geometric. Vertices are dropped randomly uniformly into a unit square and adding edges to connect any two points within a given euclidean distance. Is it planar?
- Random Erdos-Renyi. $G(n, p)$ model, n nodes are connected randomly. Each edge is included with independent probability p.
- Watts-Strogatz model. (Seen later on small world constructions).

Graphs
More complex topologies

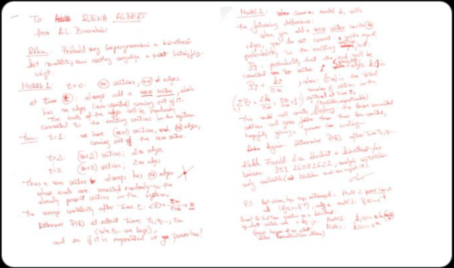
- Random geometric. Vertices are dropped randomly uniformly into a unit square and adding edges to connect any two points within a given euclidean distance. Is it planar?
- Random Erdos-Renyi. $G(n, p)$ model, n nodes are connected randomly. Each edge is included with independent probability p.
- Watts-Strogatz model. (Seen later on small world constructions).
- Barabasi-Albert model. Preferential attachment: the more connected a node is, the more likely it is to receive new links. Degree Distribution follows a power law.

Albert-László Barabási & Réka Albert

BA model was born in Porto

Albert-László Barabási
@barabasi

Happy XX Anniversary Scale-Free Model!
It was 20 years ago today that Réka and I had the email-fax exchange that resulted in the Barabasi-Albert model.
Luckily I saved the fax I sent to Reka from Porto, that, in a mixture of English and Hungarian, outlines the model.



3:14 PM · Jun 14, 2019 · Twitter Web Client

Definitions:

- A directed graph is called strongly connected if for every pair of vertices u and v there is a path from u to v and a path from v to u.

Definitions:

- A directed graph is called strongly connected if for every pair of vertices u and v there is a path from u to v and a path from v to u.
- Distance from u to v is the length of the shortest path from u to v.

Definitions:

- A directed graph is called strongly connected if for every pair of vertices u and v there is a path from u to v and a path from v to u.
- Distance from u to v is the length of the shortest path from u to v.
- A directed spanning tree with root node i is breadth first provided that each node at distance d from i in the graph appears at depth d in the tree.

Definitions:

- A directed graph is called strongly connected if for every pair of vertices u and v there is a path from u to v and a path from v to u.
- Distance from u to v is the length of the shortest path from u to v.
- A directed spanning tree with root node i is breadth first provided that each node at distance d from i in the graph appears at depth d in the tree.
- Every strongly connected graph has a breadth-first directed spanning tree.

Spanning Trees

Synchronous *SyncBFS* Algorithm

Processes communicate over directed edges. Unique UIDs are available, but network diameter and size is unknown.

Initial state in *SyncBFS*

- $parent = nil$
- $marked = False$ ($True$ in root node i_0)

Processes communicate over directed edges. Unique UIDs are available, but network diameter and size is unknown.

Initial state in *SyncBFS*

- $parent = nil$
- $marked = False$ ($True$ in root node i_0)

SyncBFS algorithm

- Process i_0 sends a *search* message in round 1.
- Unmarked processes receiving a *search* message from x do $marked = True$ and set $parent = x$, in the next round *search* messages are sent from these processes.

Complexity

- Time: At most *diam* rounds (depending on i_0 eccentricity).
- Message: $|E|$. Messages are sent across all edges E.

Complexity

- Time: At most *diam* rounds (depending on i_0 eccentricity).
- Message: $|E|$. Messages are sent across all edges E.

Child Pointers

If parents need to know their offspring, processes must reply to *search* messages with either *parent* or *nonparent*.

Complexity

- Time: At most *diam* rounds (depending on i_0 eccentricity).
- Message: $|E|$. Messages are sent across all edges E.

Child Pointers

If parents need to know their offspring, processes must reply to *search* messages with either *parent* or *nonparent*. This is only easy if the graph is undirected, but is achievable in general strongly connected graphs.

Complexity

- Time: At most *diam* rounds (depending on i_0 eccentricity).
- Message: $|E|$. Messages are sent across all edges E.

Child Pointers

If parents need to know their offspring, processes must reply to *search* messages with either *parent* or *nonparent*. This is only easy if the graph is undirected, but is achievable in general strongly connected graphs.

Termination: Making i_0 know that the tree is constructed

All processes respond with *parent* or *nonparent*. Parent terminates when all children terminate. Responses are collected from leaves to tree root.

Applications of Breath First Spanning Trees.

Aggregation (Global Computation)

Input values in each process can be aggregated towards a sync node. Each value only contributes once, many functions can be used: Sums, Averages, Max, Voting.

Applications of Breath First Spanning Trees.

Aggregation (Global Computation)

Input values in each process can be aggregated towards a sync node. Each value only contributes once, many functions can be used: Sums, Averages, Max, Voting.

Leader Election

Largest *UID* wins. All process become root of their own tree and aggregate a $Max(UID)$. Each decide by comparing their own *UID* with $Max(UID)$.

Applications of Breath First Spanning Trees.

Broadcast

Message payload m can be attached to *SyncBFS* construction ($m|E|$ message load) or broadcasted once tree is formed ($m|V|$ message load).

Carlos Baquero
DEI, FEUP, Universidade do Porto

Applications of Breath First Spanning Trees.

Broadcast

Message payload m can be attached to *SyncBFS* construction ($m|E|$ message load) or broadcasted once tree is formed ($m|V|$ message load).

Computing Diameter

Each process constructs a *SyncBFS*. Then determines *maxdist*, longest tree path. Afterwards, all processes use their trees to aggregate $max(maxdist)$ from all roots/nodes.
Complexity: Time $O(diam)$ and messages $O(diam \times |E|)$.

Asynchronous setup

Carlos Baquero
DEI, FEUP,
Universidade do Porto

We will now work on a “bare” asynchronous network model, avoiding, for now, useful abstractions like logical time and global snapshots.

System Model

Carlos Baquero
DEI, FEUP, Universidade do Porto

The lack of helping tools is compensated by a “generous” system model:

Faults

No faults.

Channels

Reliable FIFO send/receive channels.

Reliable FIFO send/receive channels
Automaton

Signature:
Input: $send(m)_{i,j}, m \in M$
Output: $receive(m)_{i,j}, m \in M$
States:
$queue = \langle\rangle$
Transitions:
$send(m)_{i,j}$
Effect:
$queue := queue + \langle m \rangle$
$receive(m)_{i,j}$
Precondition:
$queue.head = m$
Effect:
$queue.pophead()$

Reliable FIFO send/receive channels

Allowed trace behaviour

Let β be a sequence of actions, and *cause*() a function mapping each receive event $e \in \beta\mid_{receive}$ to a preceding send event $s \in \beta\mid_{send}$ such that:

1. $\forall receive(x) \in \beta\mid_{receive}: cause(receive(x)) = send(y) \Rightarrow x = y$. Messages don't come out of the blue.

Reliable FIFO send/receive channels

Allowed trace behaviour

Let β be a sequence of actions, and *cause*() a function mapping each receive event $e \in \beta\mid_{receive}$ to a preceding send event $s \in \beta\mid_{send}$ such that:

1. $\forall receive(x) \in \beta\mid_{receive}: cause(receive(x)) = send(y) \Rightarrow x = y$. Messages don't come out of the blue.
2. *cause*() is surjective. For every *send* there is a mapped *receive*. Messages are not lost.

Reliable FIFO send/receive channels
Allowed trace behaviour

Let β be a sequence of actions, and $cause()$ a function mapping each receive event $e \in \beta\mid_{receive}$ to a preceding send event $s \in \beta\mid_{send}$ such that:

1. $\forall receive(x) \in \beta\mid_{receive}: cause(receive(x)) = send(y) \Rightarrow x = y$. Messages don't come out of the blue.
2. $cause()$ is surjective. For every *send* there is a mapped *receive*. Messages are not lost.
3. $cause()$ is injective. For every *receive* there is a distinct *send*. Messages are not duplicated.

Reliable FIFO send/receive channels

Allowed trace behaviour

Let β be a sequence of actions, and *cause*() a function mapping each receive event $e \in \beta\mid_{receive}$ to a preceding send event $s \in \beta\mid_{send}$ such that:

1. $\forall receive(x) \in \beta\mid_{receive}: cause(receive(x)) = send(y) \Rightarrow x = y$. Messages don't come out of the blue.
2. *cause*() is surjective. For every *send* there is a mapped *receive*. Messages are not lost.
3. *cause*() is injective. For every *receive* there is a distinct *send*. Messages are not duplicated.
4. $receive <_\beta receive' \Rightarrow cause(receive) <_\beta cause(receive')$. Order is preserved.

Signature:

Input: $receive(''search'')_{i,j}$ $\quad j \in nbrs$

Output: $send(''search'')_{i,j}$ $\quad j \in nbrs$

Transitions:

$send(''search'')_{i,j}$

Precondition: $sendto(j) = yes$

Effect: $sendto(j) := no$

$receive(''search'')_{j,i}$

Effect:

if $i \neq i_0$ and $parent = null$ then

$parent := j$

for all $k \in nbrs \setminus \{j\}$ do

$sendto(k) := yes$

Spanning Trees

AsynchSpanningTree automaton for node i

Signature:

Input: $receive(''search'')_{i,j}$ $\quad j \in nbrs$

Output: $send(''search'')_{i,j}$ $\quad j \in nbrs$

Transitions:

$send(''search'')_{i,j}$

Precondition: $sendto(j) = yes$

Effect: $sendto(j) := no$

$receive(''search'')_{j,i}$

Effect:

if $i \neq i_0$ and $parent = null$ then

$parent := j$

for all $k \in nbrs \setminus \{j\}$ do

$sendto(k) := yes$

Channel automaton

Consumes $send(''search'')_{f,t}$ and produces $receive(''search'')_{f,t}$ in reliable FIFO order.

Carlos Baquero
DEI, FEUP, Universidade do Porto

While *AsynchSpanningTree* looks like an asynchronous translation of *SynchBFS*, the former does not necessarily produce a breadth first spanning tree.

Spanning Trees

AsynchSpanningTree vs *SynchBFS*

While *AsynchSpanningTree* looks like an asynchronous translation of *SynchBFS*, the former does not necessarily produce a breadth first spanning tree.

Faster longer paths will win over a slower direct path when setting up *parent*.

Carlos Baquero
DEI, FEUP, Universidade do Porto

While *AsynchSpanningTree* looks like an asynchronous translation of *SynchBFS*, the former does not necessarily produce a breadth first spanning tree.
Faster longer paths will win over a slower direct path when setting up *parent*.
One can however show that a spanning tree is constructed.

Invariant: A tree is gradually formed

In any reachable state, the edges defined by all *parent* variables form a spanning tree of a subgraph of G, containing i_0; moreover, if there is a message in any channel $C_{i,j}$ then i is in this spanning tree.

Carlos Baquero
DEI, FEUP, Universidade do Porto

Invariant: A tree is gradually formed

In any reachable state, the edges defined by all *parent* variables form a spanning tree of a subgraph of G, containing i_0; moreover, if there is a message in any channel $C_{i,j}$ then i is in this spanning tree.

Invariant: All contacts are searched

In any reachable state, if $i = i_0$ or $parent \neq null$, and if $j \in nbrs_i \setminus \{i_0\}$, then either $parent_j \neq null$ or $C_{i,j}$ contains a search message or $sendto(j)_i$ is yes.

Invariant: A tree is gradually formed

In any reachable state, the edges defined by all *parent* variables form a spanning tree of a subgraph of G, containing i_0; moreover, if there is a message in any channel $C_{i,j}$ then i is in this spanning tree.

Invariant: All contacts are searched

In any reachable state, if $i = i_0$ or $parent \neq null$, and if $j \in nbrs_i \setminus \{i_0\}$, then either $parent_j \neq null$ or $C_{i,j}$ contains a search message or $sendto(j)_i$ is yes.

Leading to:

Theorem

The *AsynchSpanningTree* algorithm constructs a spanning tree in the undirected graph G.

In asynchronous systems although time is unbounded it is practical to assume a upper bound on time taken to execute a process effect, time l, and time taken to deliver a message in channel, time d.

In asynchronous systems although time is unbounded it is practical to assume a upper bound on time taken to execute a process effect, time l, and time taken to deliver a message in channel, time d.

Complexity

- Messages are $O(|E|)$.
- Time is $O(diam(l + d))$.

Spanning Trees
AsynchSpanningTree

In asynchronous systems although time is unbounded it is practical to assume a upper bound on time taken to execute a process effect, time l, and time taken to deliver a message in channel, time d.

Complexity

- Messages are $O(|E|)$.
- Time is $O(diam(l + d))$.

Altough a tree with height h, such that $h > diam$, can occur it only occurs if it does not take more time than a tree with $h = diam$.

Carlos Baquero
DEI, FEUP, Universidade do Porto

In asynchronous systems although time is unbounded it is practical to assume a upper bound on time taken to execute a process effect, time l, and time taken to deliver a message in channel, time d.

Complexity

- Messages are $O(|E|)$.
- Time is $O(diam(l + d))$.

Altough a tree with height h, such that $h > diam$, can occur it only occurs if it does not take more time than a tree with $h = diam$.
Faster long paths must be faster!

Carlos Baquero
DEI, FEUP, Universidade do Porto

Child pointers and Broadcast

If nodes report *parent* or *nonparent* one can build a tree that broadcasts.

Carlos Baquero
DEI, FEUP, Universidade do Porto

Child pointers and Broadcast

If nodes report *parent* or *nonparent* one can build a tree that broadcasts.
Is the time complexity of this tree still $O(diam(l + d))$?

Carlos Baquero
DEI, FEUP, Universidade do Porto

Child pointers and Broadcast

If nodes report *parent* or *nonparent* one can build a tree that broadcasts.
Is the time complexity of this tree still $O(diam(l+d))$?
No, because a fast path is not always fast. Complexity is $O(h(l+d))$, at most $O(n(l+d))$, where $n = |V|$.

Broadcast with Acks

Its is possible to build a *AsynchBcastAck* algorithm that collects acknowledgements as the tree is constructed. Upon incoming broadcast messages each node Acks if they already know the broadcast and Acks to parent once when all neighbours Ack to them.

Spanning Trees
AsynchSpanningTree

Child pointers and Broadcast

If nodes report *parent* or *nonparent* one can build a tree that broadcasts.
Is the time complexity of this tree still $O(diam(l + d))$?
No, because a fast path is not always fast. Complexity is $O(h(l + d))$, at most $O(n(l + d))$, where $n = |V|$.

Broadcast with Acks

Its is possible to build a *AsynchBcastAck* algorithm that collects acknowledgements as the tree is constructed. Upon incoming broadcast messages each node Acks if they already know the broadcast and Acks to parent once when all neighbours Ack to them.

Leader Election with *AsynchBcastAck*

This algorithm includes termination and if all nodes initiate it and report their UIDs it can be used for Leader Election with unknown diameter and number of nodes.

Epidemic Broadcast Trees

Plumtree protocol

- Gossip broadcast
  - + highly scalable and resilient
  - - excessive message overhead
- Tree-based broadcast
  - + small message complexity
  - - fragile in the presence of failures

Can we get the best of both worlds?

- Eager push. Nodes immediately forward new messages
- Pull. Nodes periodically query for new messages
- Lazy push. Nodes push new message ids and accept pulls

In lazy push, there is a separation among payload and metadata

Epidemic Broadcast Trees

Gossiping into tree

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Nodes sample random peers into a eagerPush set
- Neighbours should be stable and TCP can be used
- Links are kept reciprocal (towards undirected graph)
- First message reception puts origin in eagerPush
- Further duplicate receptions moves source to lazyPush
- Eager push of payload and lazy push of metadata

Epidemic Broadcast Trees

Repairing a broken tree

If the tree breaks, and graph stays connected, nodes get metadata but not payloads. This is detected by timer expiration and the metadata source in lazyPush is moved to eagerPush. Potential redundancy (due to cycles) is cleared later by the standard algorithm.

Small Worlds

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Milgram experiment "Six degrees of separation".
- Path lengths were calculated on sequences of letter forwardings.

Small Worlds

- Milgram experiment "Six degrees of separation".
- Path lengths were calculated on sequences of letter forwardings.
- In theory, letters would be send from random senders and progressively forwarded to random recipients.
- At each point the letter was forwarded to an address and recipient more likely to know the final destination recipient.

Small Worlds

- Milgram experiment "Six degrees of separation".
- Path lengths were calculated on sequences of letter forwardings.
- In theory, letters would be send from random senders and progressively forwarded to random recipients.
- At each point the letter was forwarded to an address and recipient more likely to know the final destination recipient.
- An average path length close to 6 hops was found in the results.

Small Worlds

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Milgram experiment "Six degrees of separation".
- Path lengths were calculated on sequences of letter forwardings.
- In theory, letters would be send from random senders and progressively forwarded to random recipients.
- At each point the letter was forwarded to an address and recipient more likely to know the final destination recipient.
- An average path length close to 6 hops was found in the results.
- The question was: "Why should there exist short chains of acquaintances linking together arbitrary pairs of strangers?"

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.
- Are random graphs a good model for people acquaintances?

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.
- Are random graphs a good model for people acquaintances?
- Not so, since people's graphs have more clustering. If A is friend to B and C, then it is likely that B and C are also friends.

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.
- Are random graphs a good model for people acquaintances?
- Not so, since people's graphs have more clustering. If A is friend to B and C, then it is likely that B and C are also friends.
- Watts and Strogatz proposed a model that mixes short range and long range contacts.

Random graphs and clustering

Carlos Baquero
DEI, FEUP, Universidade do Porto

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.
- Are random graphs a good model for people acquaintances?
- Not so, since people's graphs have more clustering. If A is friend to B and C, then it is likely that B and C are also friends.
- Watts and Strogatz proposed a model that mixes short range and long range contacts.
- Nodes establish k local contacts using some distance metric among vertices (say in a ring or lattice) and then a few long range contacts uniformly at random.

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices. See Erdős–Rényi model.
- The resulting random graph is known to depict a low diameter and thus could support small paths. $O(\log n)$.
- Are random graphs a good model for people acquaintances?
- Not so, since people's graphs have more clustering. If A is friend to B and C, then it is likely that B and C are also friends.
- Watts and Strogatz proposed a model that mixes short range and long range contacts.
- Nodes establish k local contacts using some distance metric among vertices (say in a ring or lattice) and then a few long range contacts uniformly at random.
- Resulting in low diameter and high clustering.

Routing in Small Worlds

Carlos Baquero
DEI, FEUP, Universidade do Porto

- If we flood a Watts and Strogatz graph we will stumble on a short route between to arbitrary points. A global observer could also pinpoint a $O(\log N)$ path.
- But, can we pick a path with local knowledge and a distance metric?

Routing in Small Worlds

- If we flood a Watts and Strogatz graph we will stumble on a short route between to arbitrary points. A global observer could also pinpoint a $O(\log N)$ path.
- But, can we pick a path with local knowledge and a distance metric?
- Not so easy. Going to the next nearest point does not home in the target, we can keep jumping and only achieve $O(\sqrt{N})$ paths. These paths lack locallity.

Routing in Small Worlds

- If we flood a Watts and Strogatz graph we will stumble on a short route between to arbitrary points. A global observer could also pinpoint a $O(\log N)$ path.
- But, can we pick a path with local knowledge and a distance metric?
- Not so easy. Going to the next nearest point does not home in the target, we can keep jumping and only achieve $O(\sqrt{N})$ paths. These paths lack locallity.
- Kleinberg solved this issue by choosing a probability function that can restore locality to long links. (Check in R: `n=10; s = exp(log(n)* (runif(1000) -1)); hist(s,100)`.)

Routing in Small Worlds

- If we flood a Watts and Strogatz graph we will stumble on a short route between to arbitrary points. A global observer could also pinpoint a $O(\log N)$ path.
- But, can we pick a path with local knowledge and a distance metric?
- Not so easy. Going to the next nearest point does not home in the target, we can keep jumping and only achieve $O(\sqrt{N})$ paths. These paths lack locallity.
- Kleinberg solved this issue by choosing a probability function that can restore locality to long links. (Check in R: `n=10; s = exp(log(n)* (runif(1000) -1)); hist(s,100).`)
- Long range contacts can be tuned to become more clustered in the vicinity. The target is to have uniformity across all distance scales, a property found in DHT designs like Chord, and locally find $O(\log^2 N)$ routes.

Uniformity across distance scales

Pieter Bruegel the Elder - The Hunters in the Snow

Carlos Baquero
DEI, FEUP, Universidade do Porto