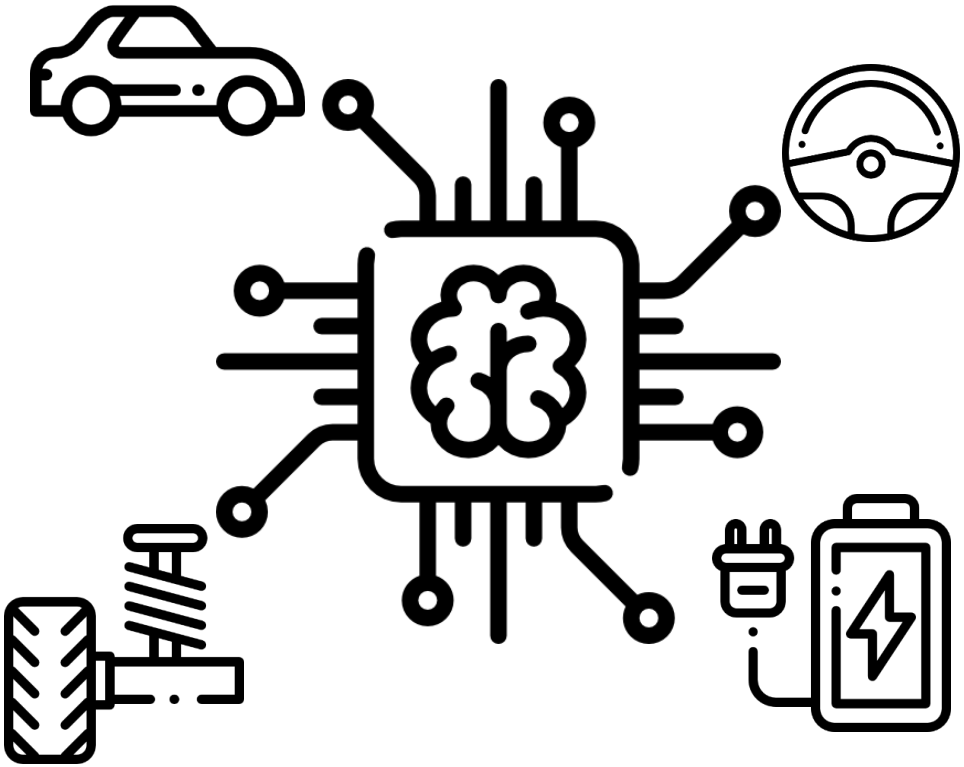
Institute of Automotive Technology
TUM School of Engineering and Design
Technical University of Munich

Artificial Intelligence in Automotive Technology

Maximilian Geißlinger / Fabian Netzler

Prof. Dr.-Ing. Markus Lienkamp

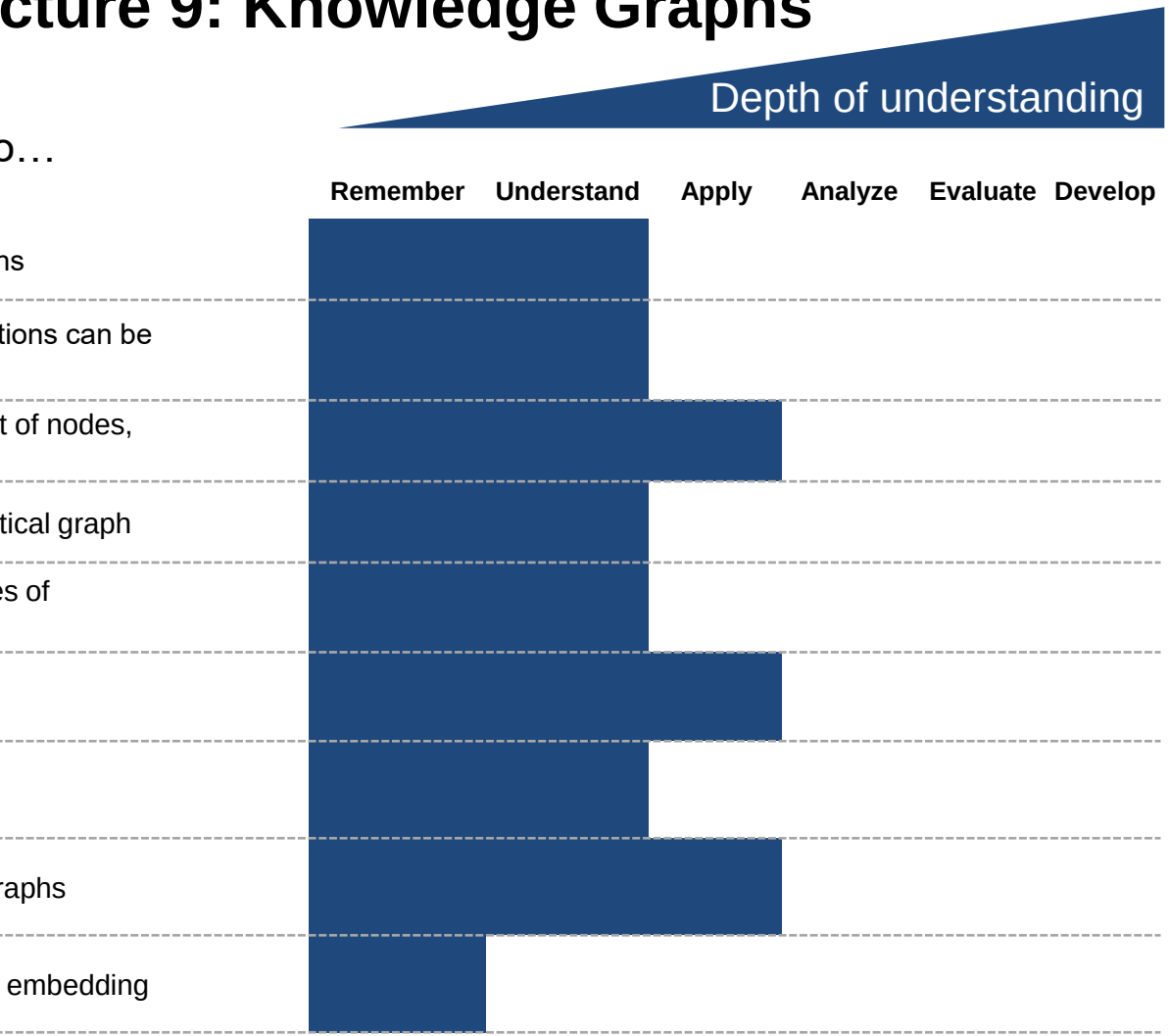


Lecture Overview

Lecture 16:15-17:45 | Practice 17:45-18:30	
1 Introduction: Artificial Intelligence	20.10.2022 – Maximilian Geißlinger
2 Perception	27.10.2022 – Sebastian Huber
3 Supervised Learning: Regression	03.11.2022 – Fabian Netzler
4 Supervised Learning: Classification	10.11.2022 – Andreas Schimpe
5 Unsupervised Learning: Clustering	17.11.2022 – Andreas Schimpe
6 Introduction: Artificial Neural Networks	24.11.2022 – Lennart Adenaw
7 Deep Neural Networks	08.12.2022 – Domagoj Majstorovic
8 Convolutional Neural Networks	15.12.2022 – Domagoj Majstorovic
9 Knowledge Graphs	12.01.2023 – Fabian Netzler
10 Recurrent Neural Networks	19.01.2023 – Matthias Rowold
11 Reinforcement Learning	26.01.2023 – Levent Ögretmen
12 AI-Development	02.02.2023 – Maximilian Geißlinger
13 Guest Lecture	09.02.2023 – to be announced

Objectives for Lecture 9: Knowledge Graphs

After the lecture you are able to…



Knowledge Graphs: From Data to Wisdom
Prof. Dr. Markus Lienkamp

(Fabian Netzler, M. Sc.)

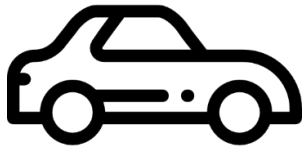
Agenda

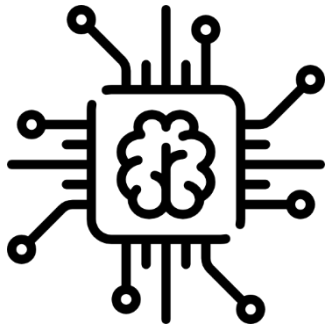
1. Chapter: Introduction into Graphs

2. Chapter: Path Planning
    - 2.1 Depth/Breadth Search
    - 2.2 Dijkstra

3. Chapter: Learning over Property Graphs
    - 3.1 Node Embedding
    - 3.2 Graph Neural Networks

4. Chapter: Summary



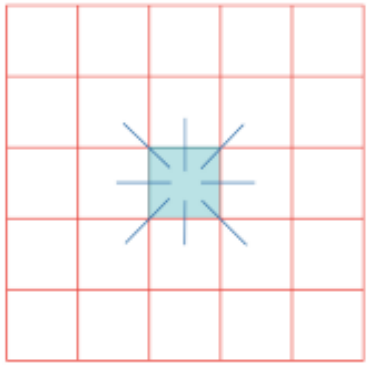


Hype Cycle for Artificial Intelligence, 2020

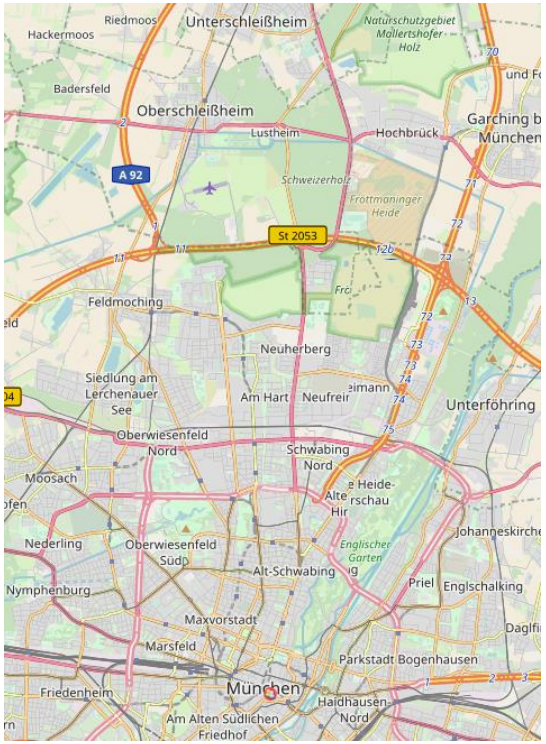
Source: Gartner

Introduction

Representing Data Though Graphs

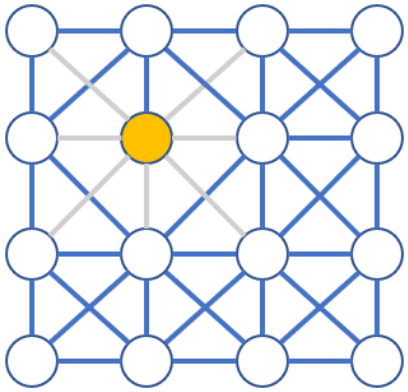


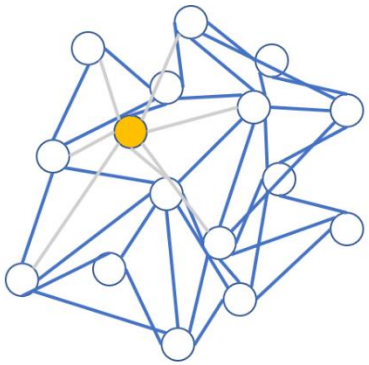
https://www.mos.ed.tum.de/ftm/forschungsfelder/intelligente-fahrzeugsysteme/indy-autonomous-challenge/



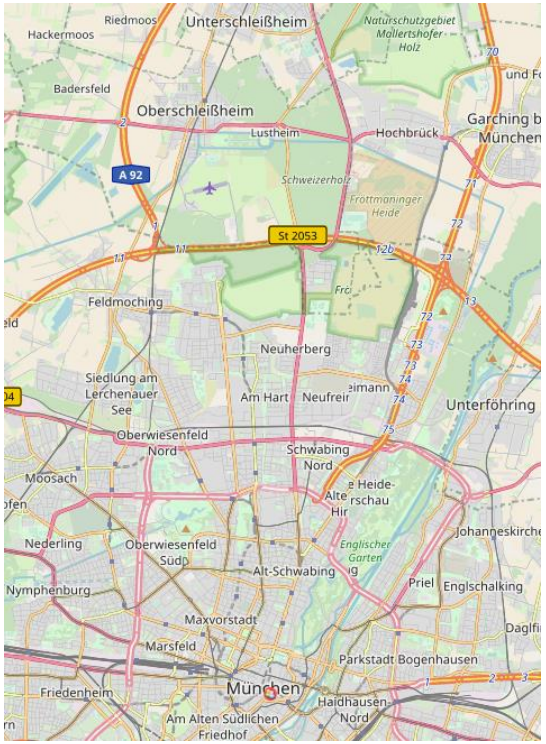
Introduction

Representing Data Though Graphs



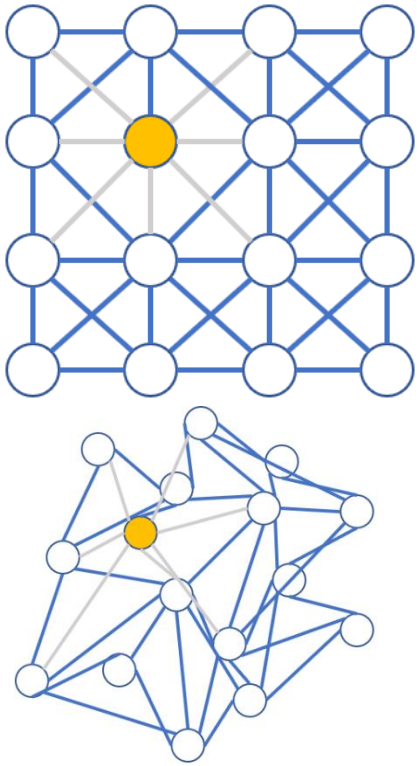


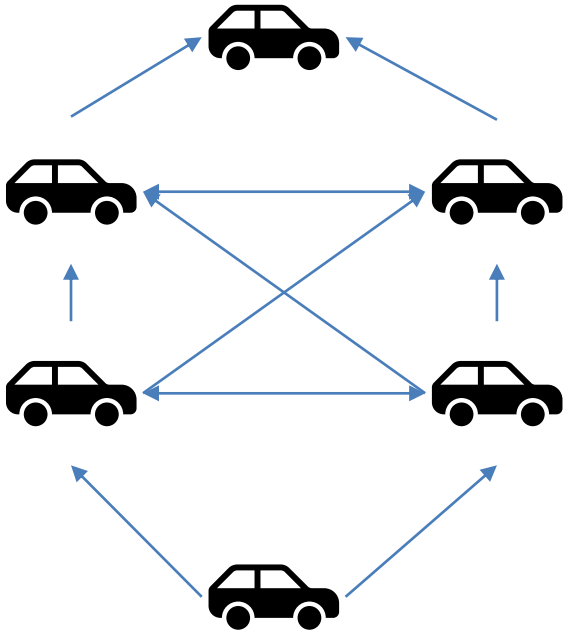
https://www.mos.ed.tum.de/ftm/forschungsfelder/intelligente-fahrzeugsysteme/indy-autonomous-challenge/

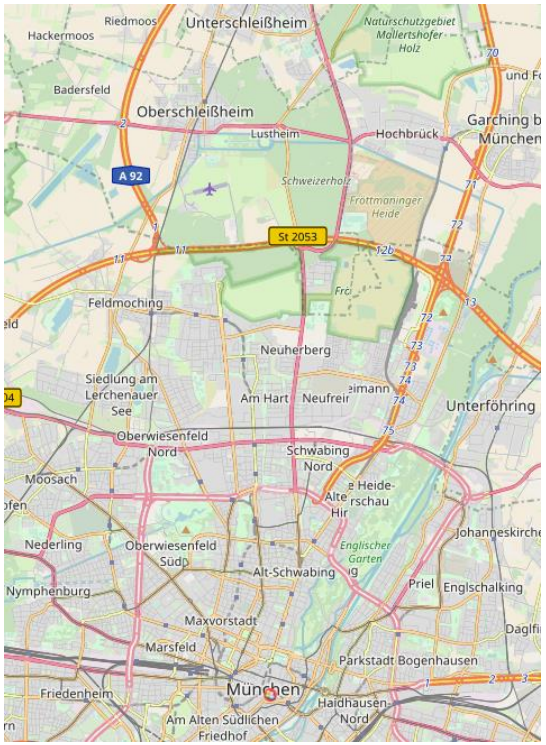


Introduction

Representing Data Though Graphs

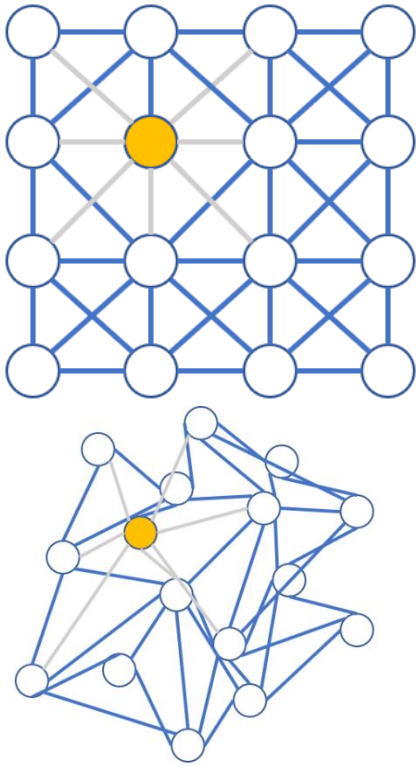


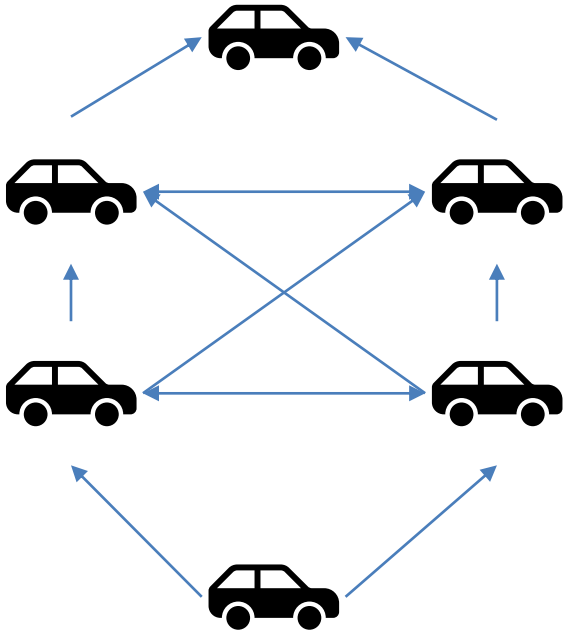


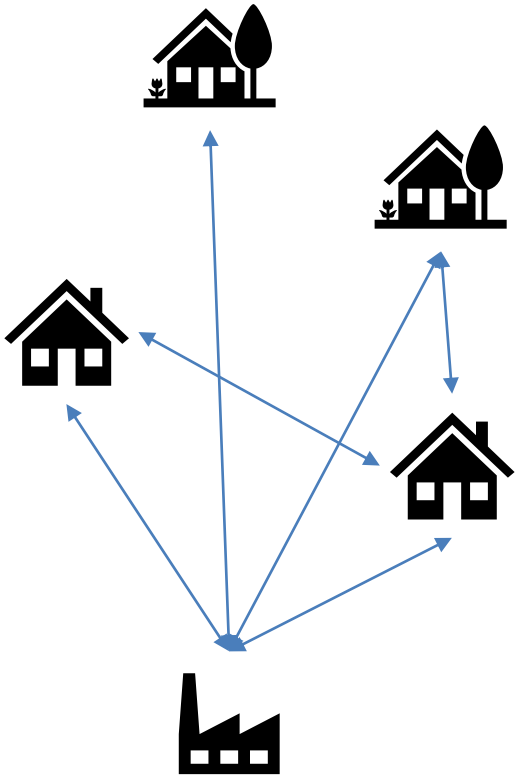


Introduction

Representing Data Though Graphs

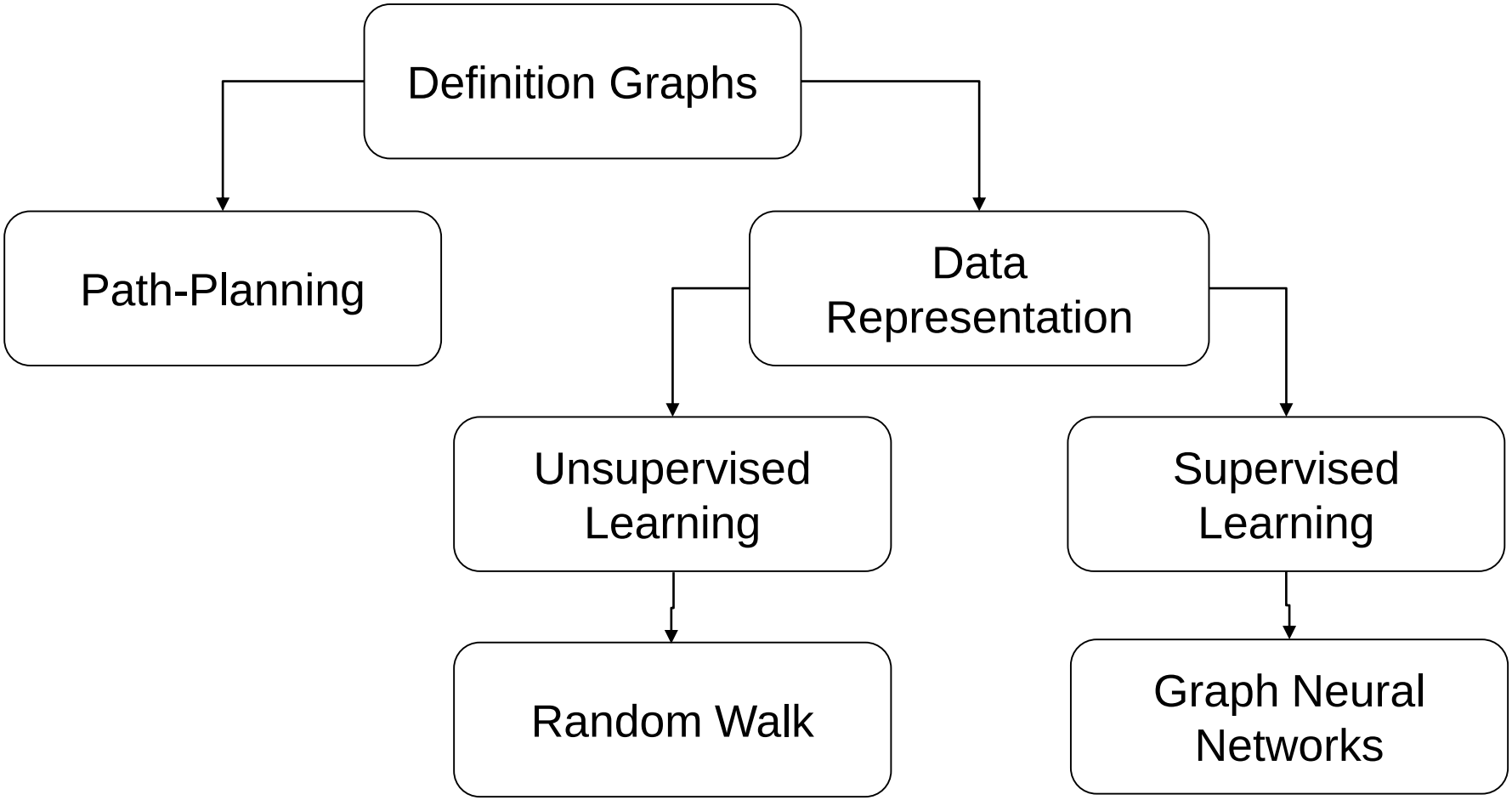






Introduction

Overview and Placement in the Lecture



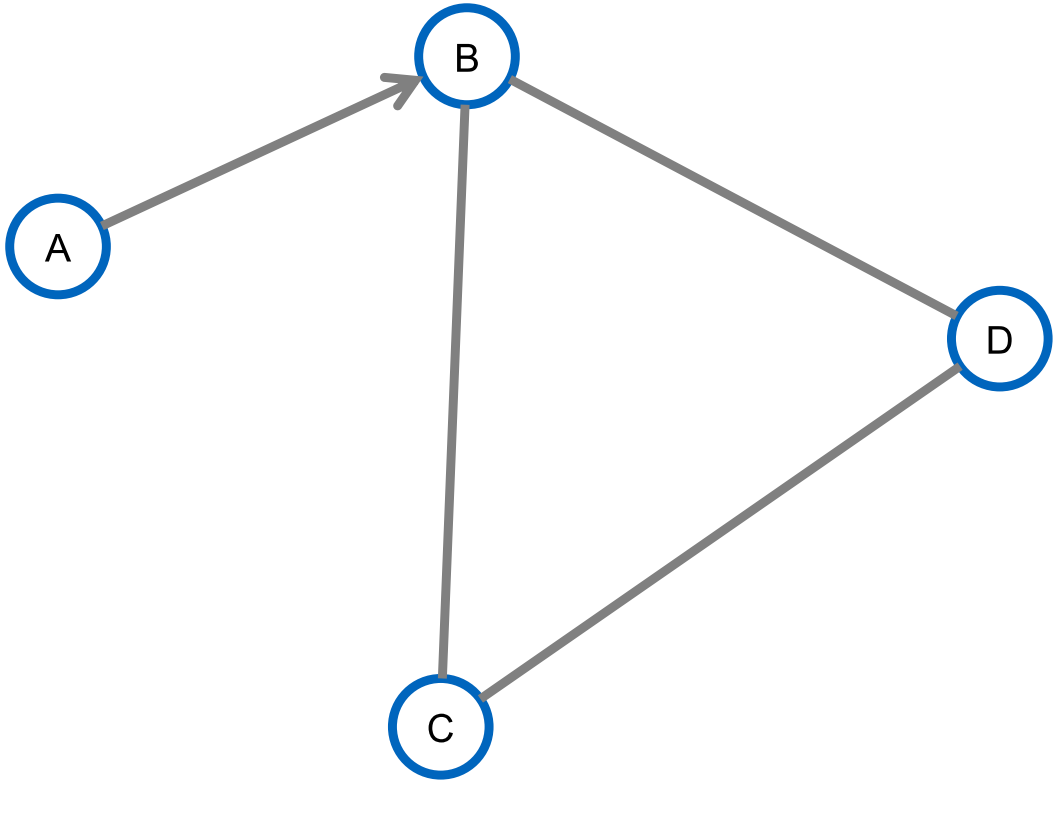
Introduction – Graphs and Path Planning

Graphs – Basic Elements

Node

Directed Edge

Undirected Edge



Introduction – Graphs and Path Planning

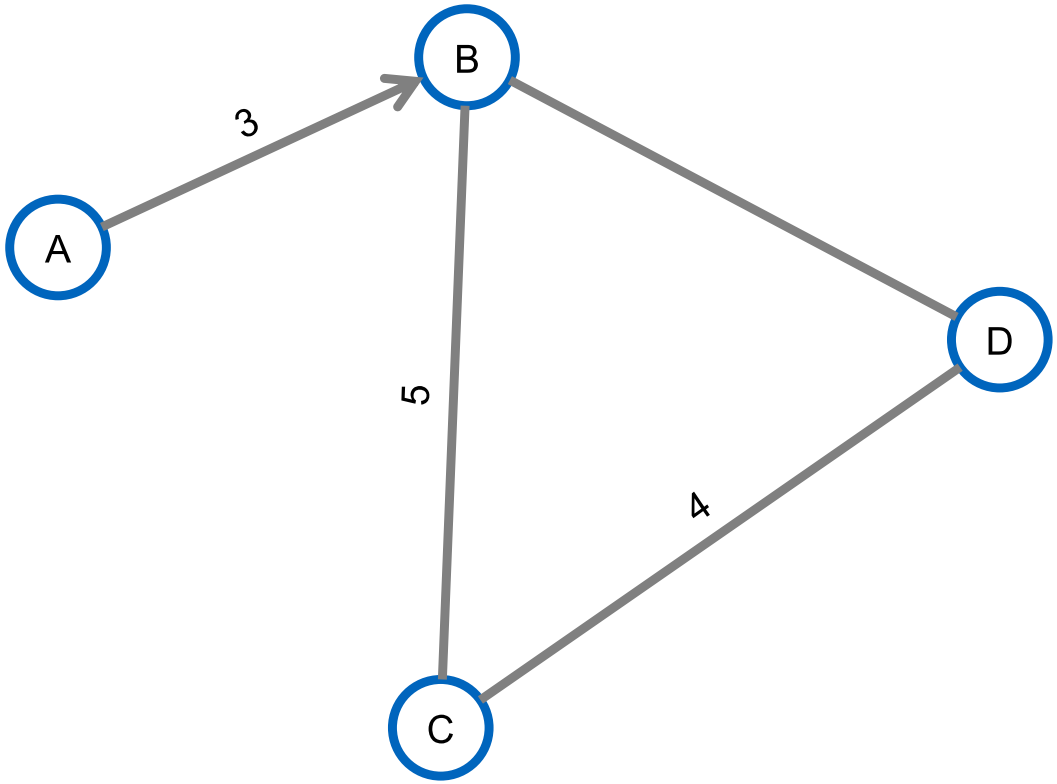
Graphs – Elements as Sets

Weighted Edge

3

Unweighted Edge

Edge weights are interpretable as **cost** (e.g. time, distance …)



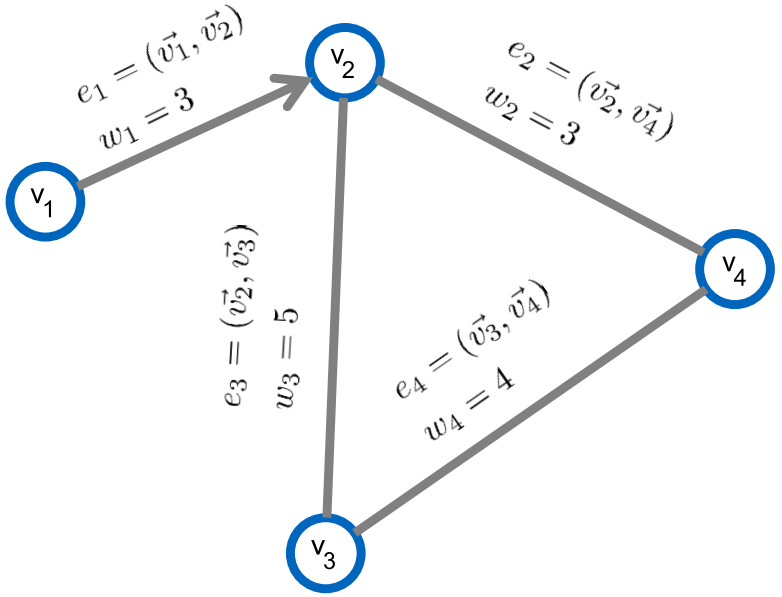
Introduction

Graphs – Formal Description

$G:$	$Graph$
$V:$	$Nodes$
$\vec{v}:$	$Node$
$E:$	$Edges$
$e:$	$Edge$
$W:$	$Weights$
$w:$	$Weight$

$$G = (V, E, W)$$

$$e = (\vec{u}, \vec{v});\ \vec{u}, \vec{v} \in V$$



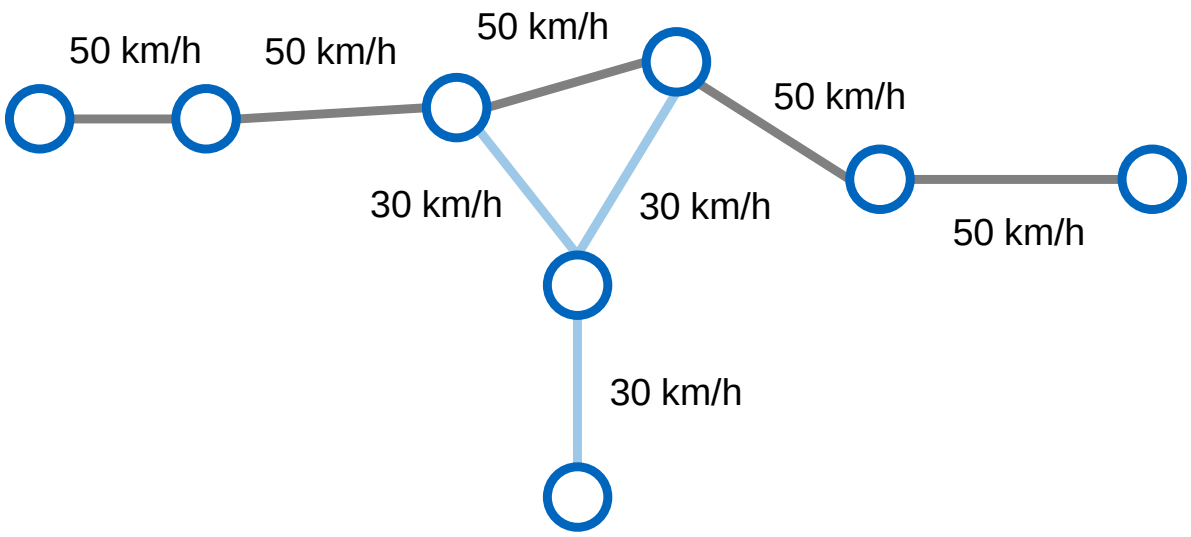
$$E = \{e_1, e_2, e_3\}$$
$$V = \{\vec{v_1}, \vec{v_2}, \vec{v_3}, \vec{v_4}\}$$
$$W = \{w_1, w_2, w_3, w_4\}$$

Introduction – Graphs and Path Planning

From OSM to Routable Graphs

Initial Graph

OSM Data provides **geometry** and **speed limits**

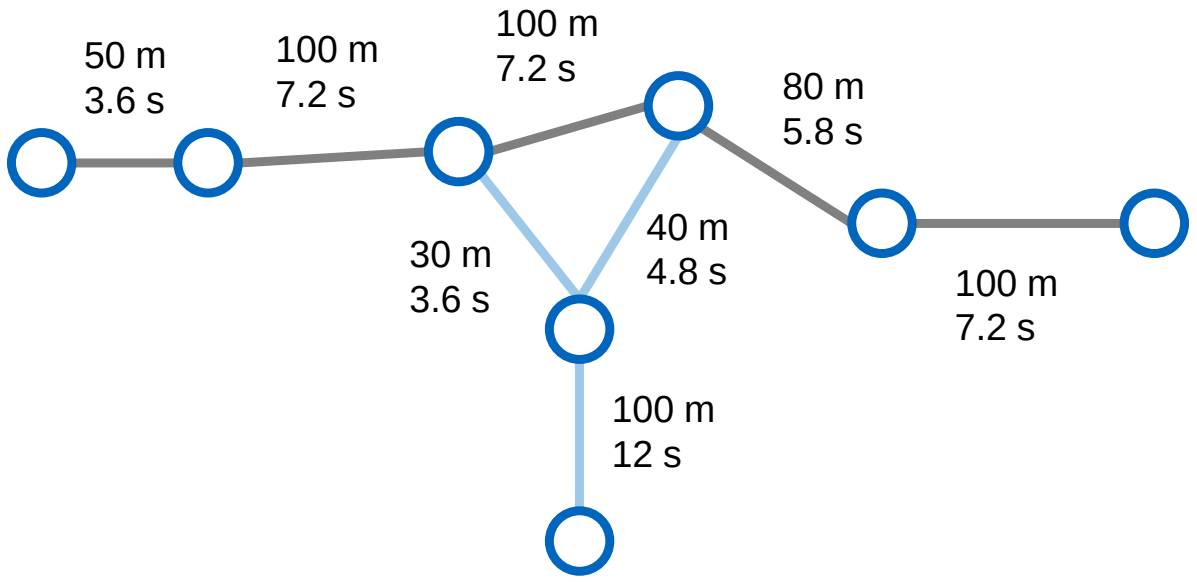


Introduction – Graphs and Path Planning

From OSM to Routable Graphs

Augmented Graph

Edge weights like **distances** and **timespans** are calculated

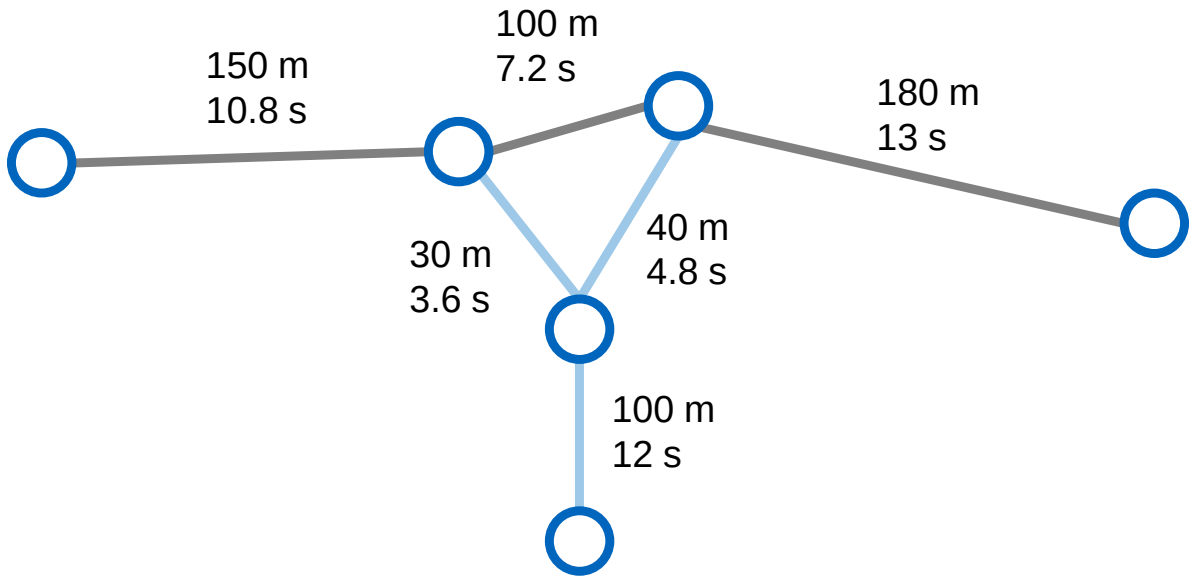


Introduction – Graphs and Path Planning

From OSM to Routable Graphs

Compressed Graph

Graph compression **reduces complexity**:
Reduction of computational effort

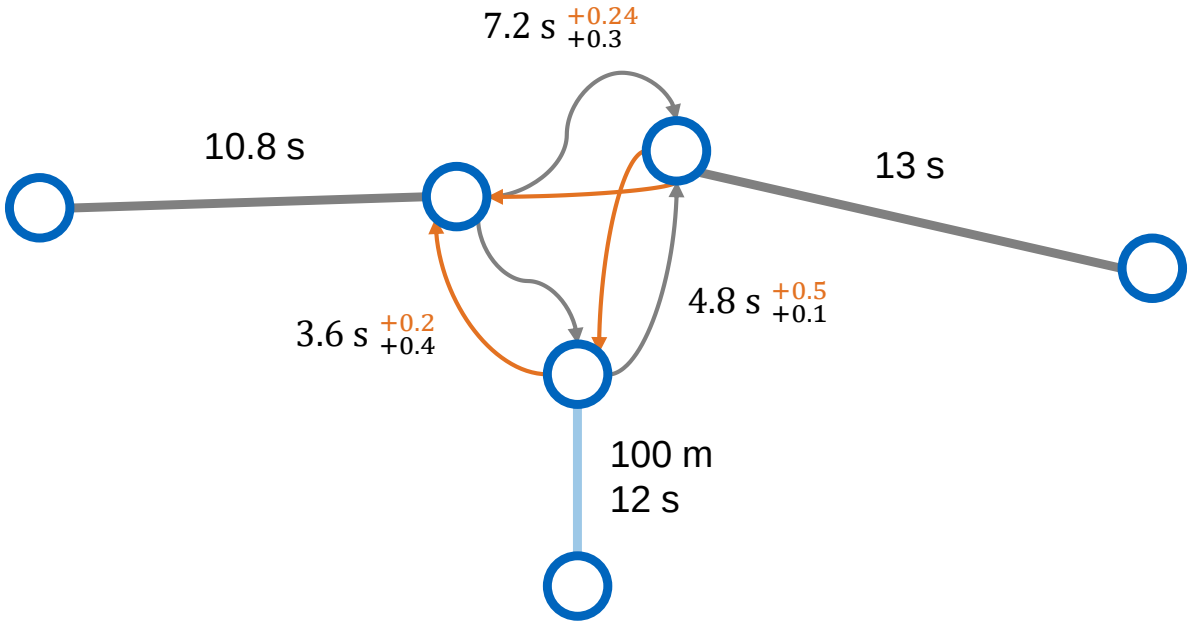


Introduction – Graphs and Path Planning

From OSM to Routable Graphs

Compressed Graph with turn restrictions and turn penalties

Turn restrictions are derived from OSM tags, **turn penalties** may be introduced for time optimal routing



Introduction – Digital Maps

Lecture Assumptions

Assumption	Description
$w_i \geq 0 \, \forall w_i \in W$	Only **non-negative edge weights**
$e = (\vec{v}, \vec{u}) = (\vec{u}, \vec{v}) \, \forall \, \vec{u}, \vec{v} \in V, e \in E$ $w = w((\vec{v}, \vec{u})) = w((\vec{u}, \vec{v}))$	**Edges** are **undirected**
$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}, \vec{v} \in V$	Nodes are given by their coordinates in a **two dimensional space**
$w(\vec{v}, \vec{u}) = d(\vec{v}, \vec{u}) = \|\|\vec{v} - \vec{u}\|\|_2$ $\|\|\vec{v} - \vec{u}\|\|_2 = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$	**Edge weights** are defined by a distance function. The **euclidean distance** is used

Knowledge Graphs: From Data to Wisdom
Prof. Dr. Markus Lienkamp

(Fabian Netzler, M. Sc.)

Agenda

1. Chapter: Introduction into Graphs

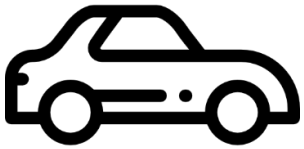
2. Chapter: Path Planning

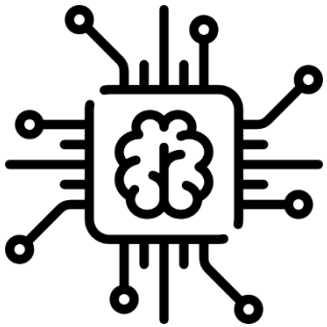
- 2.1 Depth/Breadth Search
- 2.2 Dijkstra

3. Chapter: Learning over Property Graphs

- 3.1 Node Embedding
- 3.2 Graph Neural Networks

4. Chapter: Summary



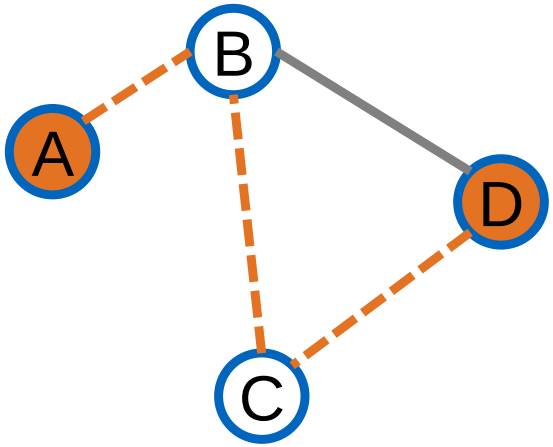


Algorithms

Pathfinding: Problem Formulations

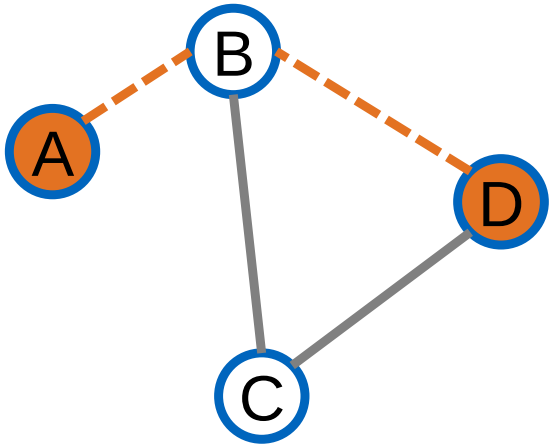
A : Find a Path

- Depth First
- Breadth First
- (Best First)



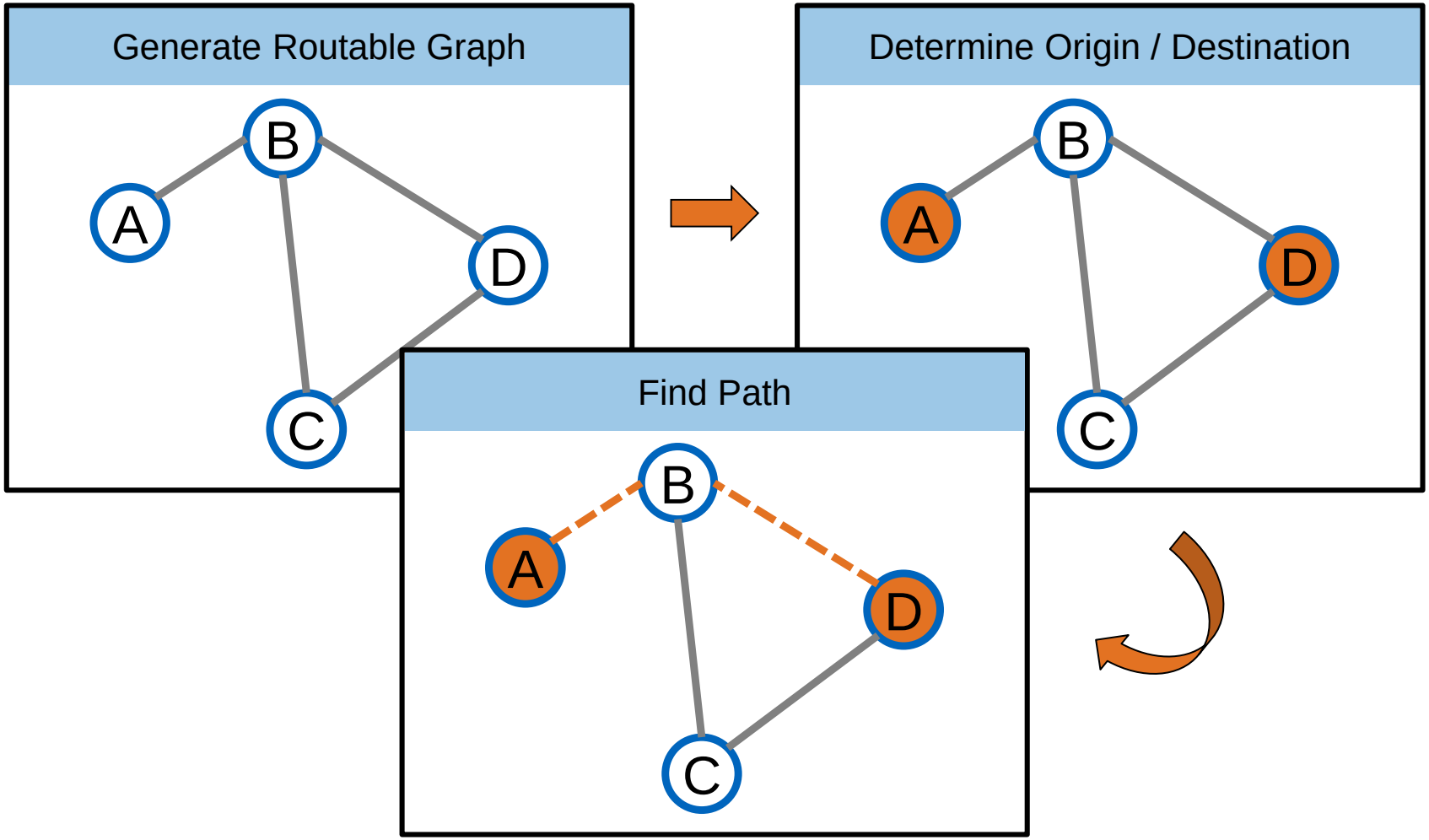
B : Find an optimal Path

- Dijkstra
- A*



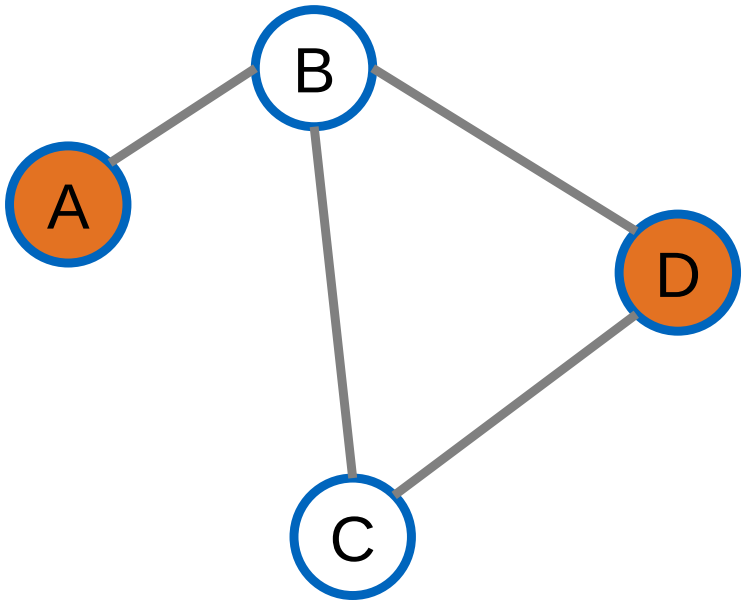
Algorithms

Overview Routing

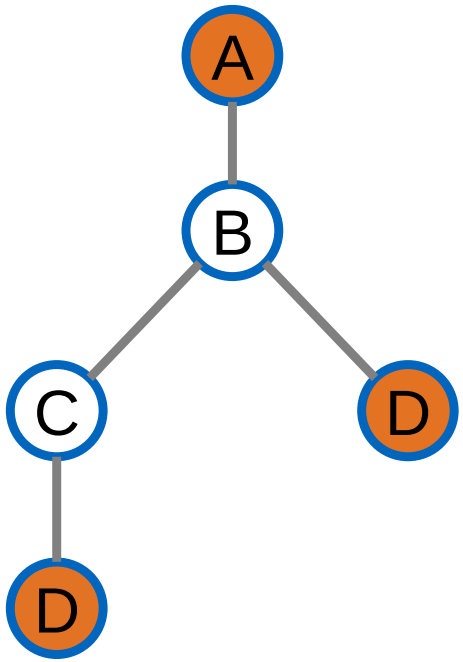


Algorithms
Search Trees

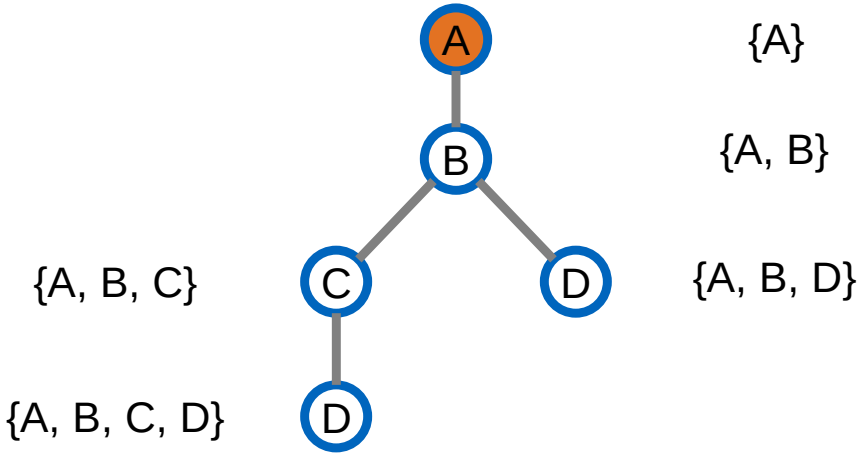
Graph



Search Tree



Algorithms
Search Trees



A search tree can be made from a graph

Each child node denotes a path that is a one-step extension of the path denoted by its parent

Converting graphs into search trees:
Tracing out all possible paths until no further extension is possible

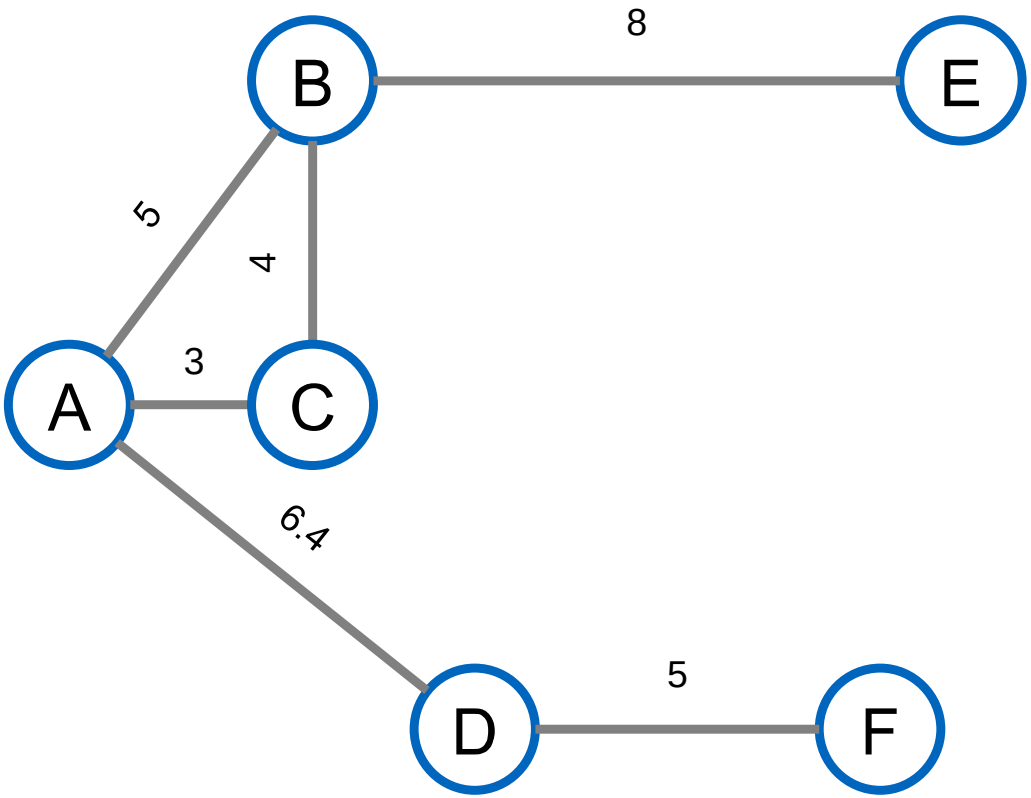
Algorithms
Completeness

A search algorithm is complete if it is guaranteed to find an existing solution in a finite amount of time.

All algorithms in this lecture

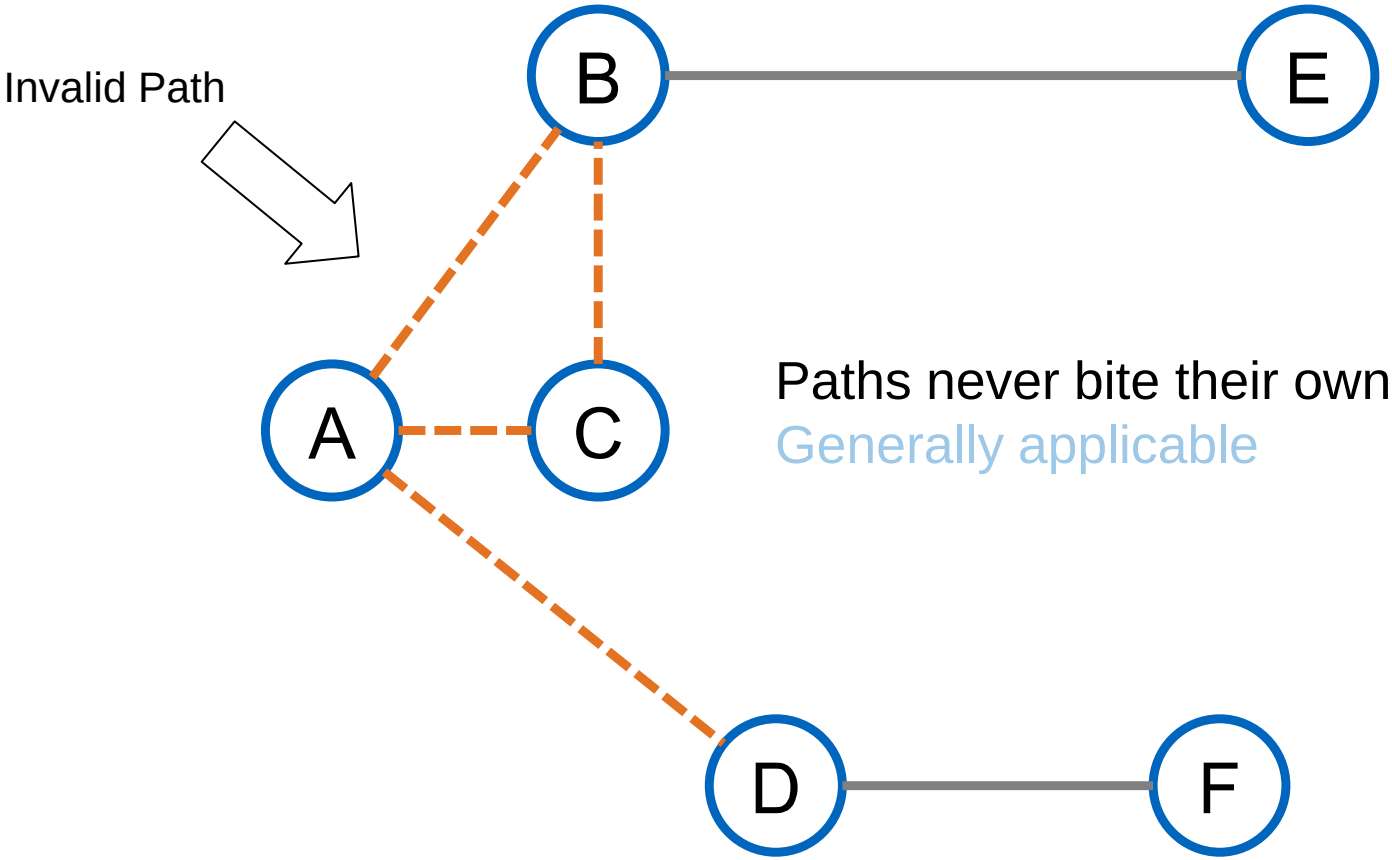
Algorithms

Pathfinding Example Graph



Algorithms

Ground Rules Example Graph

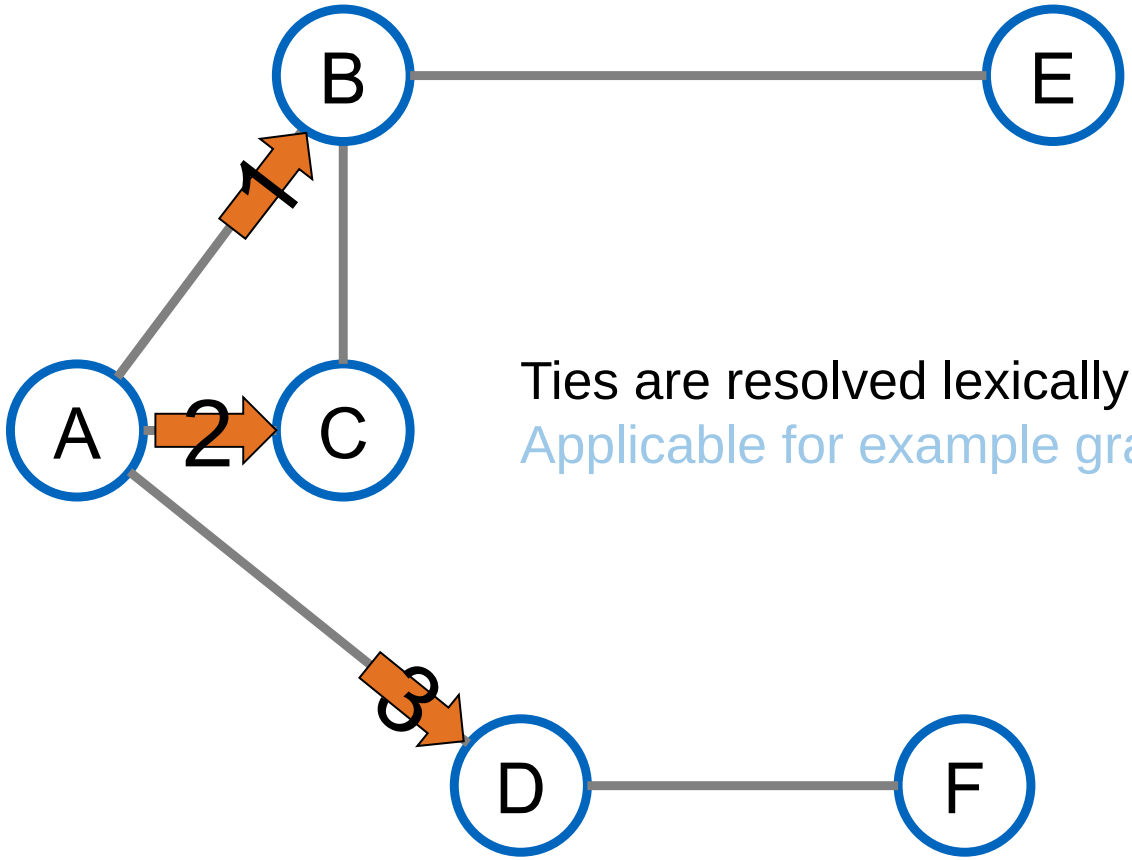


Paths never bite their own tail

Generally applicable

Algorithms

Ground Rules Example Graph



Ties are resolved lexically

Applicable for example graph

Algorithms – Depth First

Description

Algorithm:
From „Artificial Intelligence“ by Patrick H. Winston

- Form a one-element queue consisting of a zero-length path that contains only the root node

---

- Until the first path in the queue terminates at the goal node or the queue is empty,
  - Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the front of the queue

- If the goal node is found, announce success; otherwise announce failure

Algorithms – Depth First

Description

Algorithm:
From „Artificial Intelligence“ by Patrick H. Winston

- Form a one-element queue consisting of a zero-length path that contains only the root node

---

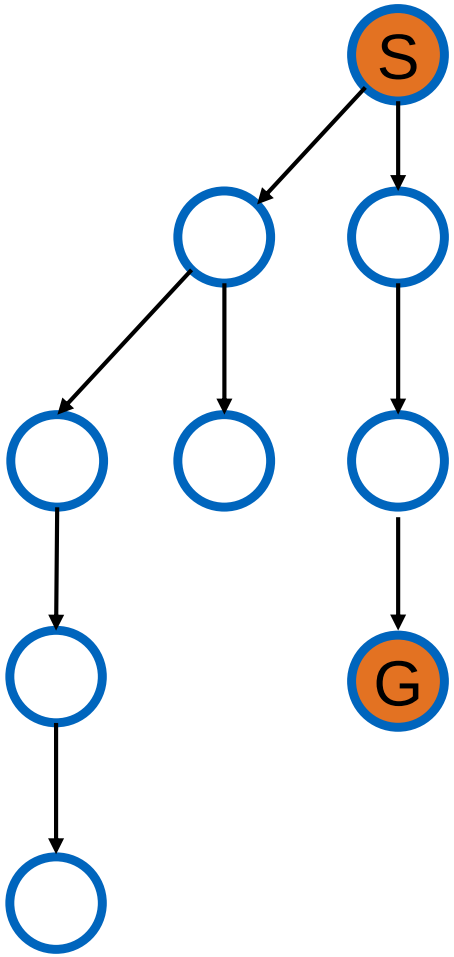
- Until the first path in the queue is empty,
  - Remove the first path from the queue, create new paths by extending the first path to all neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the front of the queue

- If the goal node is found, announce success; otherwise announce failure

Diving into the Search Tree

Algorithms – Depth First

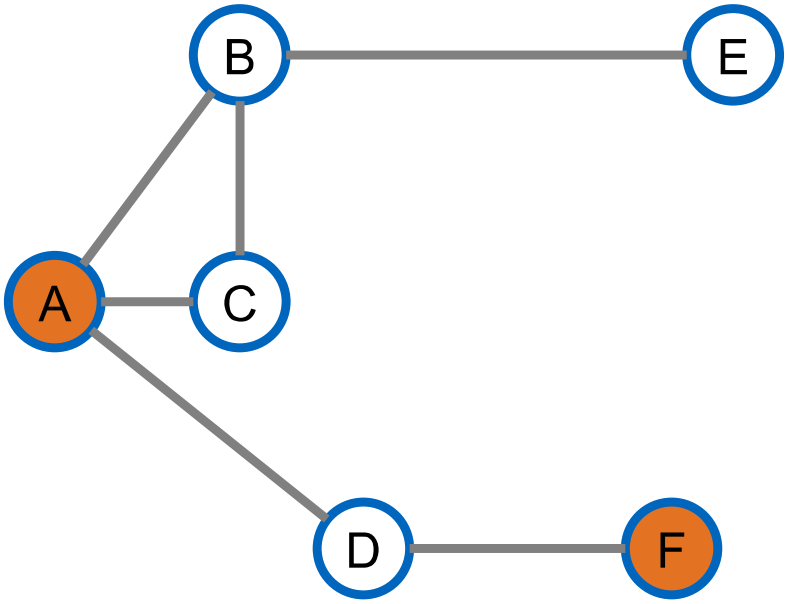
Characteristic Search Tree



narrow and deep

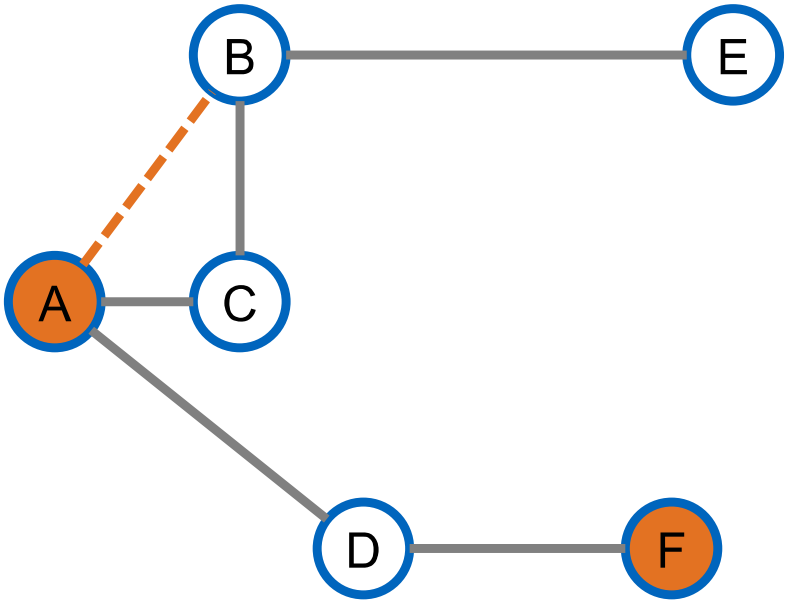
Algorithms – Depth First

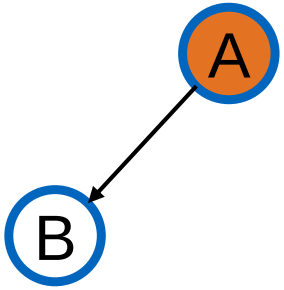
Pathfinding Example Graph



Algorithms – Depth First

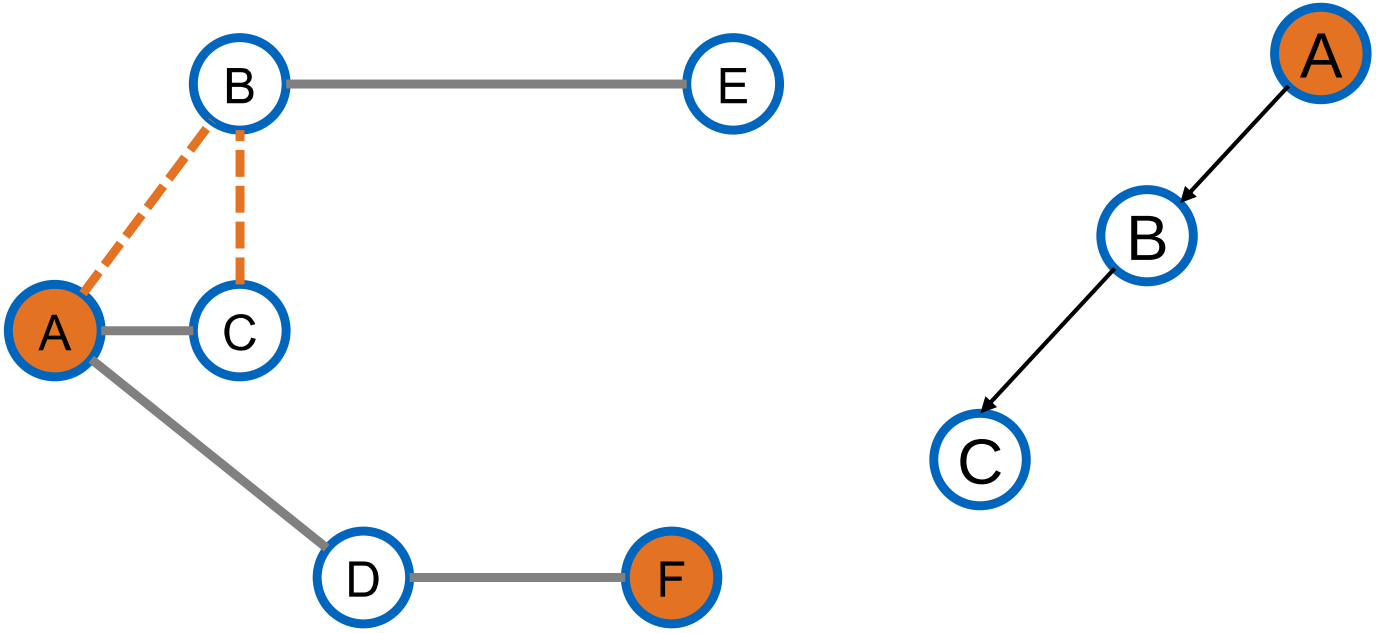
Pathfinding Example Graph





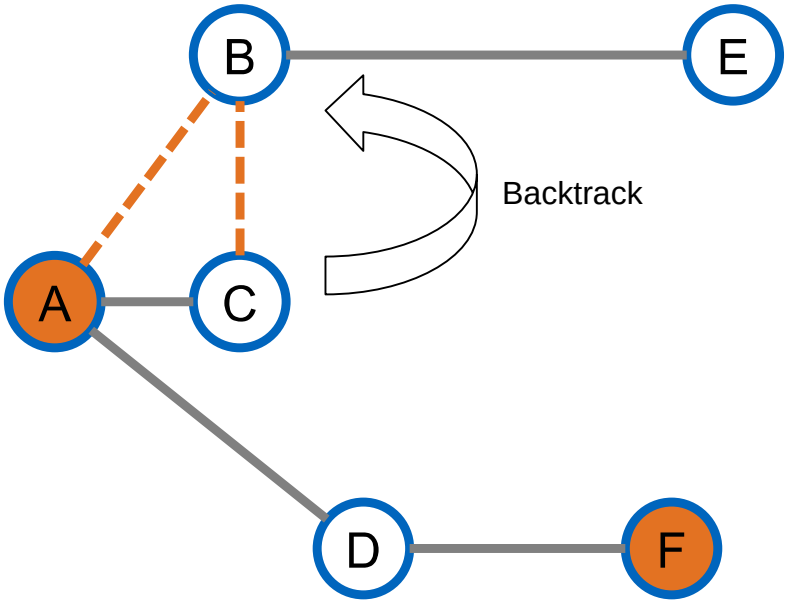
Algorithms – Depth First

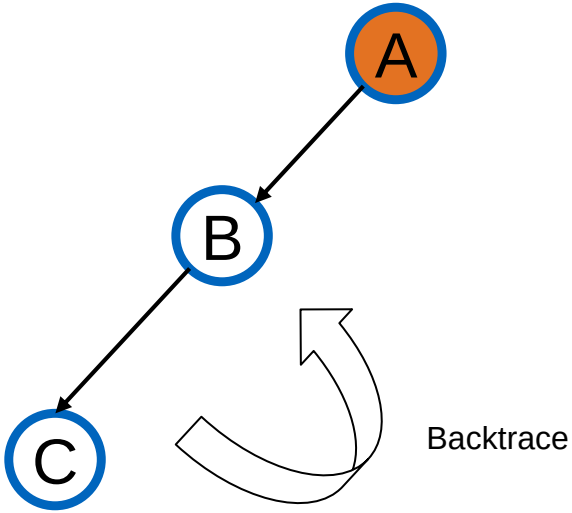
Pathfinding Example Graph



Algorithms – Depth First

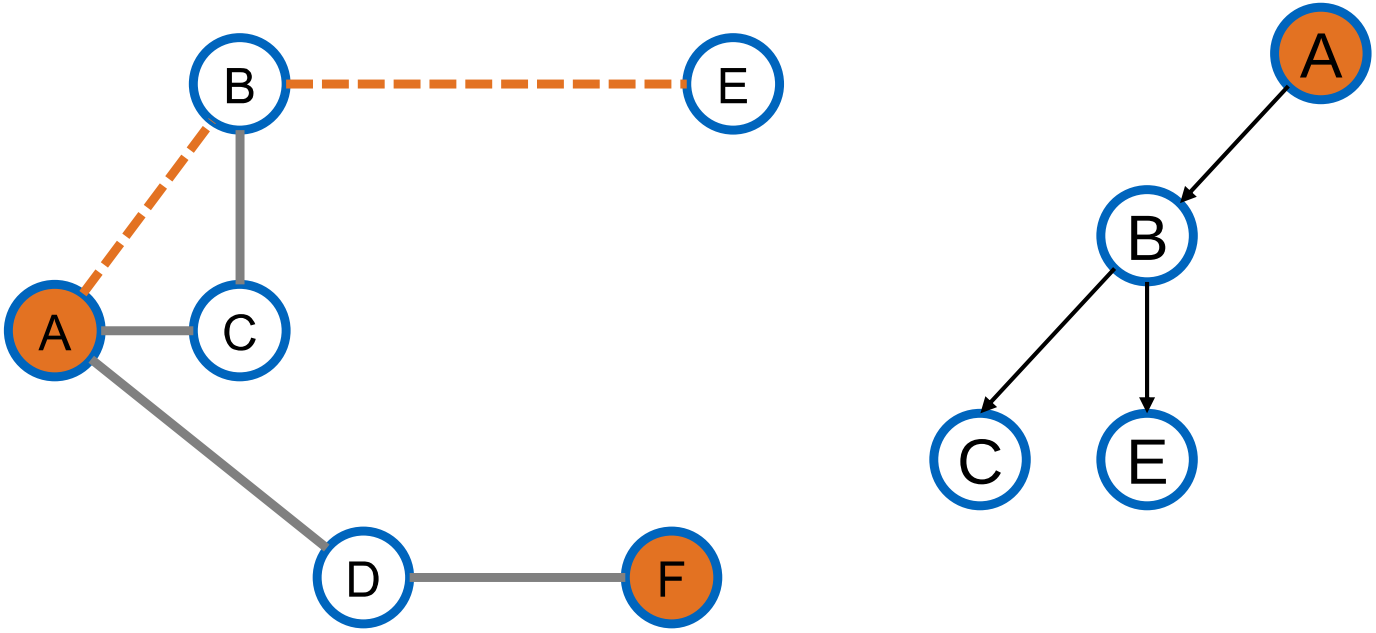
Pathfinding Example Graph





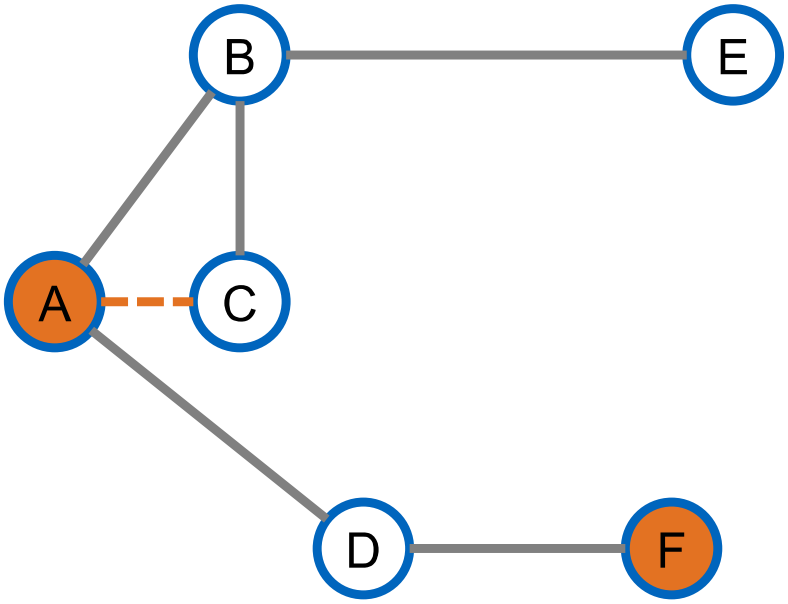
Algorithms – Depth First

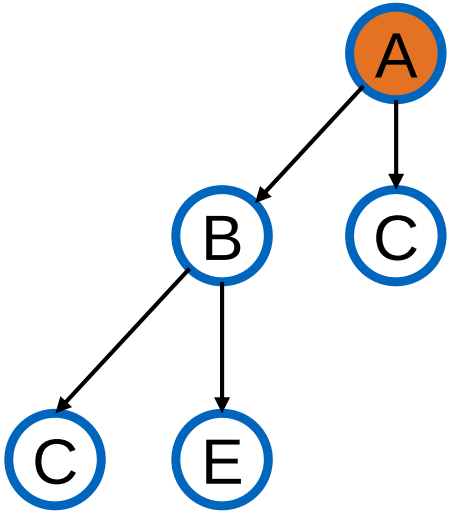
Pathfinding Example Graph



Algorithms – Depth First

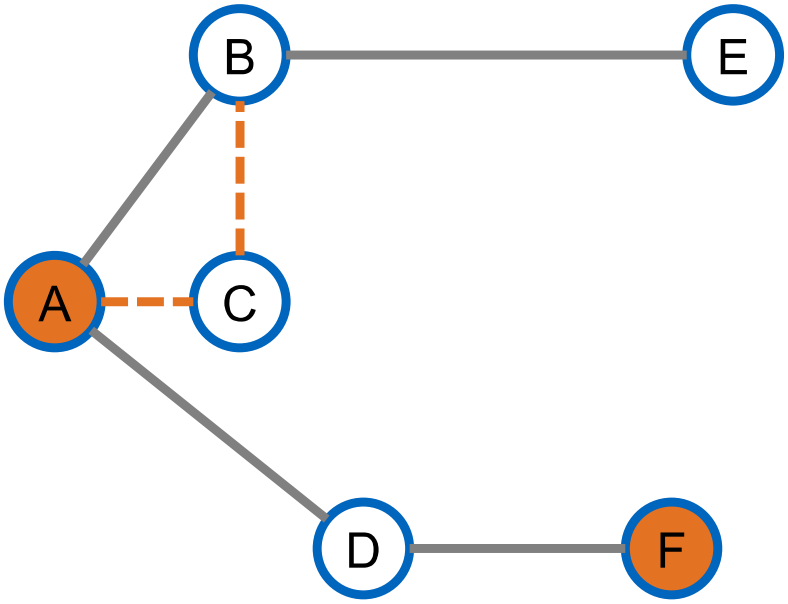
Pathfinding Example Graph

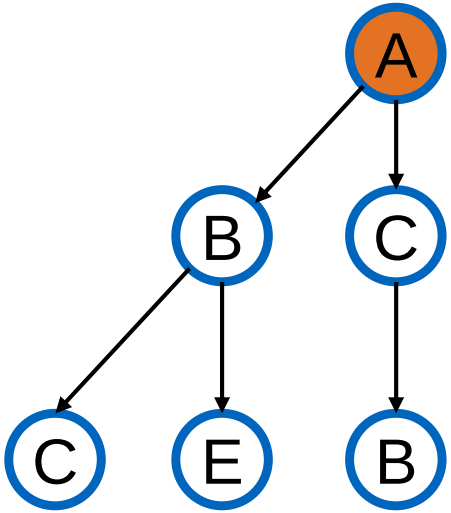




Algorithms – Depth First

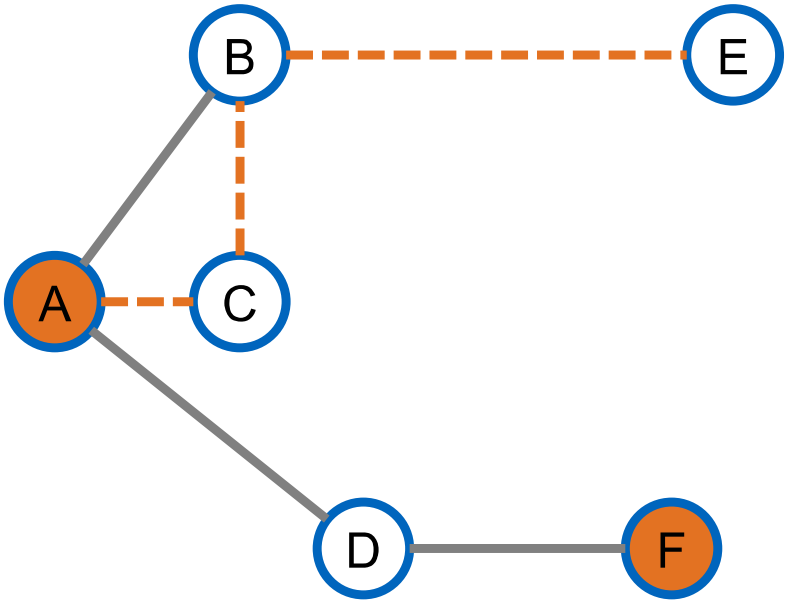
Pathfinding Example Graph

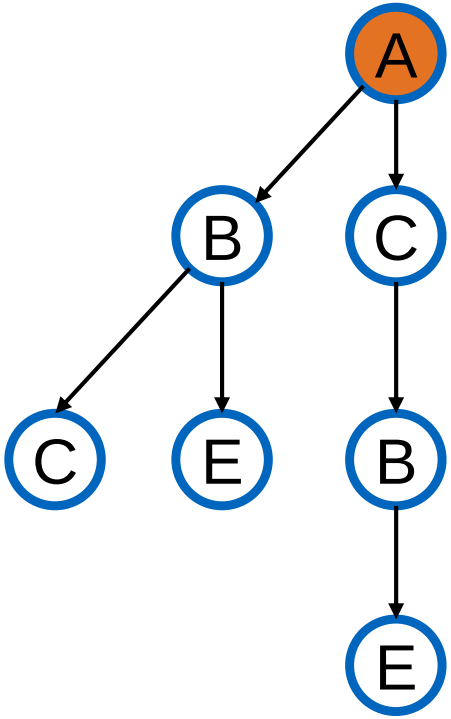




Algorithms – Depth First

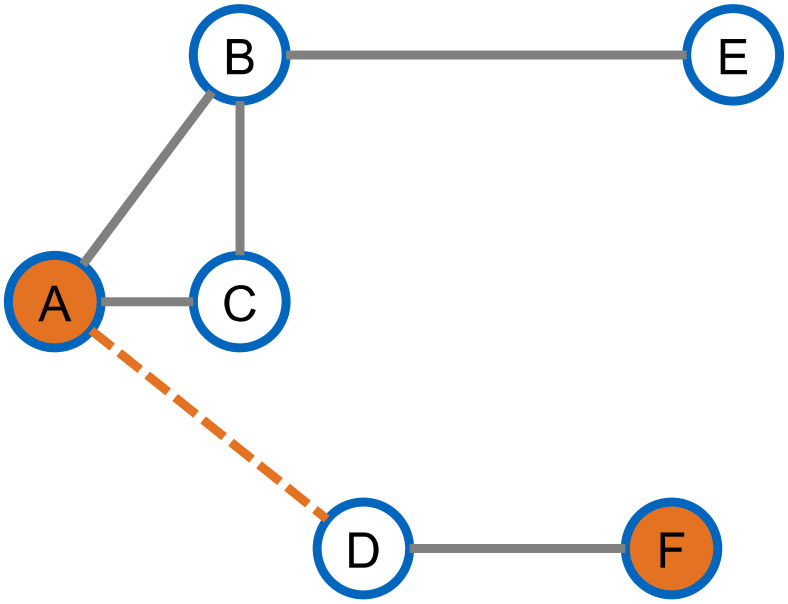
Pathfinding Example Graph

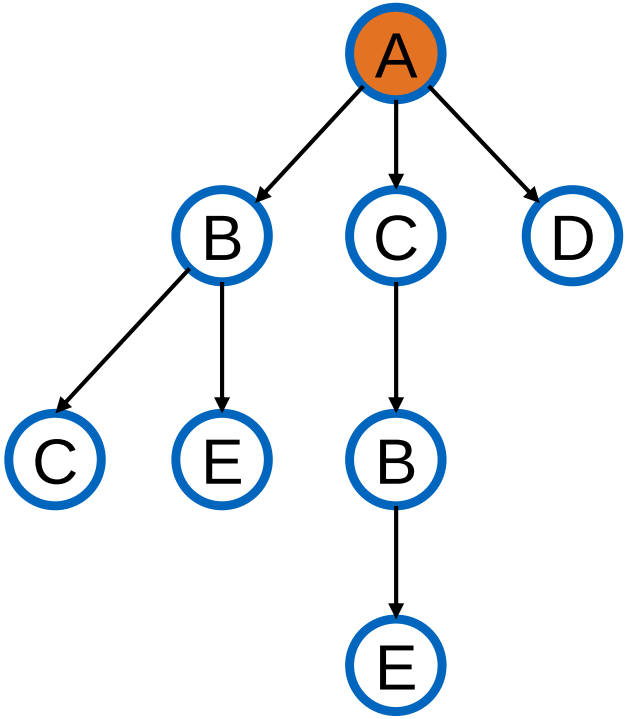




Algorithms – Depth First

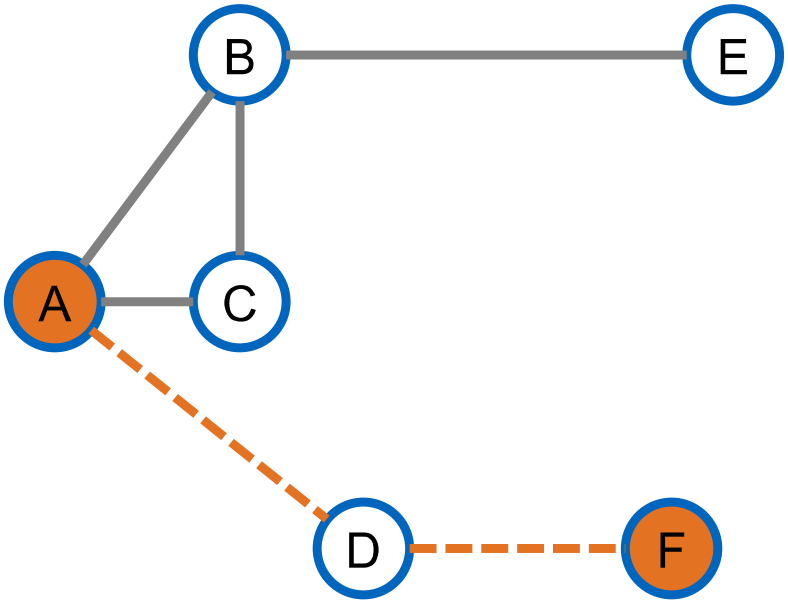
Pathfinding Example Graph

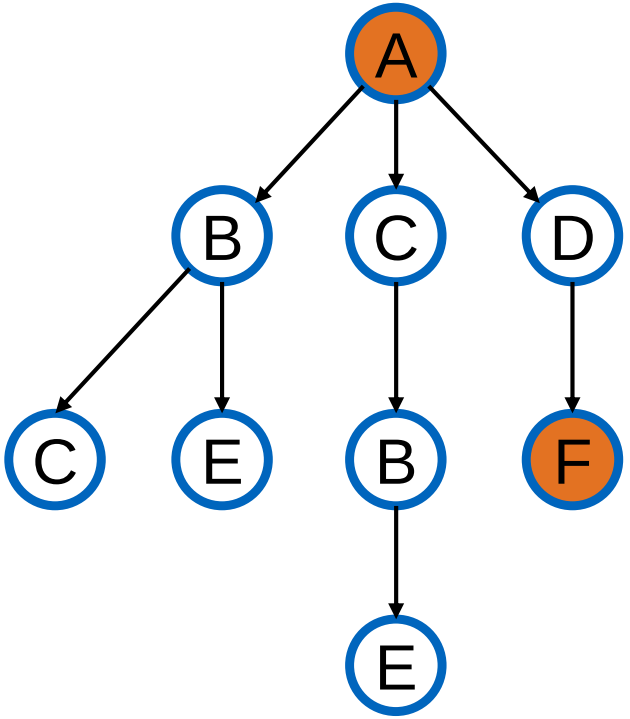




Algorithms – Depth First

Pathfinding Example Graph





Algorithms – Breadth First

Description

Algorithm:
From „Artificial Intelligence“ by Patrick H. Winston

- Form a one-element queue consisting of a zero-length path that contains only the root node

---

- Until the first path in the queue terminates at the goal node or the queue is empty,
  - Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the back of the queue

- If the goal node is found, announce success; otherwise announce failure

Algorithms – Breadth First

Description

Algorithm:
From „Artificial Intelligence“ by Patrick H. Winston

- Form a one-element queue consisting of a zero-length path that contains only the root node

---

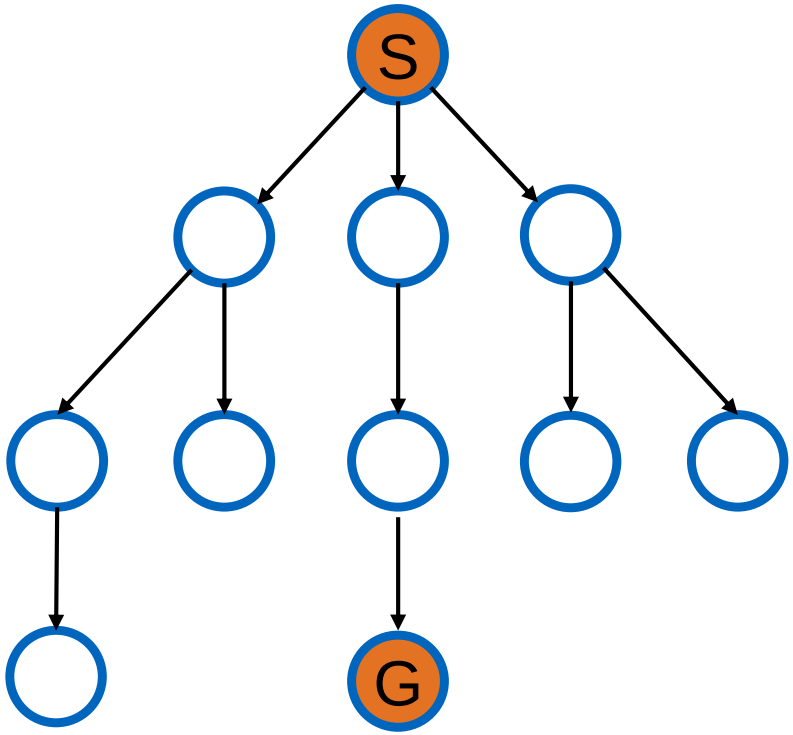
- Until the first path in the queue ends at the goal node or the queue is empty,
  - Remove the first path from the queue, create new paths by extending the first path to all neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the back of the queue

- If the goal node is found, announce success; otherwise announce failure

Scanning the Search Tree Level by Level

Algorithms – Breadth First

Characteristic Search Tree

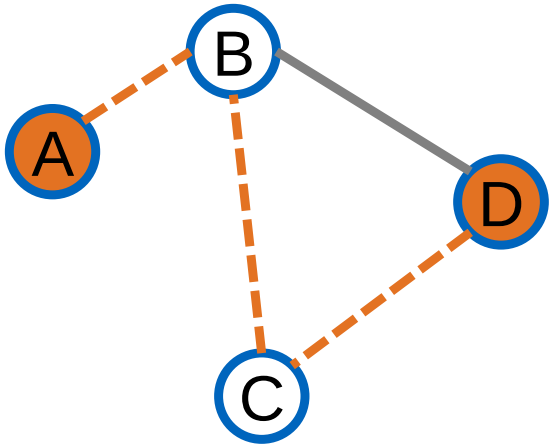


wide and shallow

Pathfinding: Problem Formulations

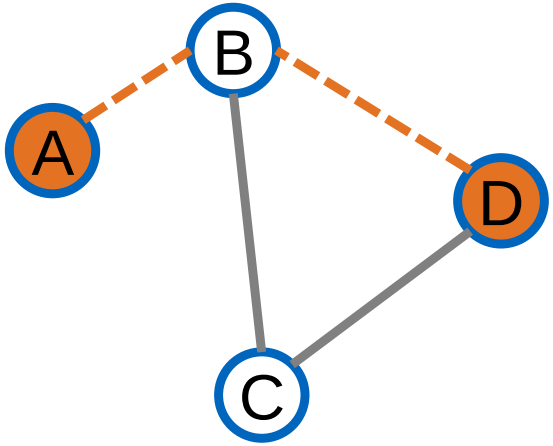
A : Find a Path

- Depth First
- Breadth First
- (Best First)



B : Find an optimal Path

- Dijkstra
- A*



Pathfinding – Dijkstra

Description

Algorithm:

- Form a one-element queue consisting of a zero-length path that contains only the root node

Core Idea: Next Slide

- If the goal node is found, announce success; otherwise announce failure

Pathfinding – Dijkstra
Description

Algorithm – Core Idea:

- Until the first path in the queue terminates at the goal node or the queue is empty,
  - Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node
  - Reject all new paths with loops
  - If two or more paths reach a common node, delete all those paths except the one that reaches the common node with the minimum cost.
  - Add the new paths, if any, to the queue
  - Sort all paths by their accumulated costs

Pathfinding – Dijkstra

Description

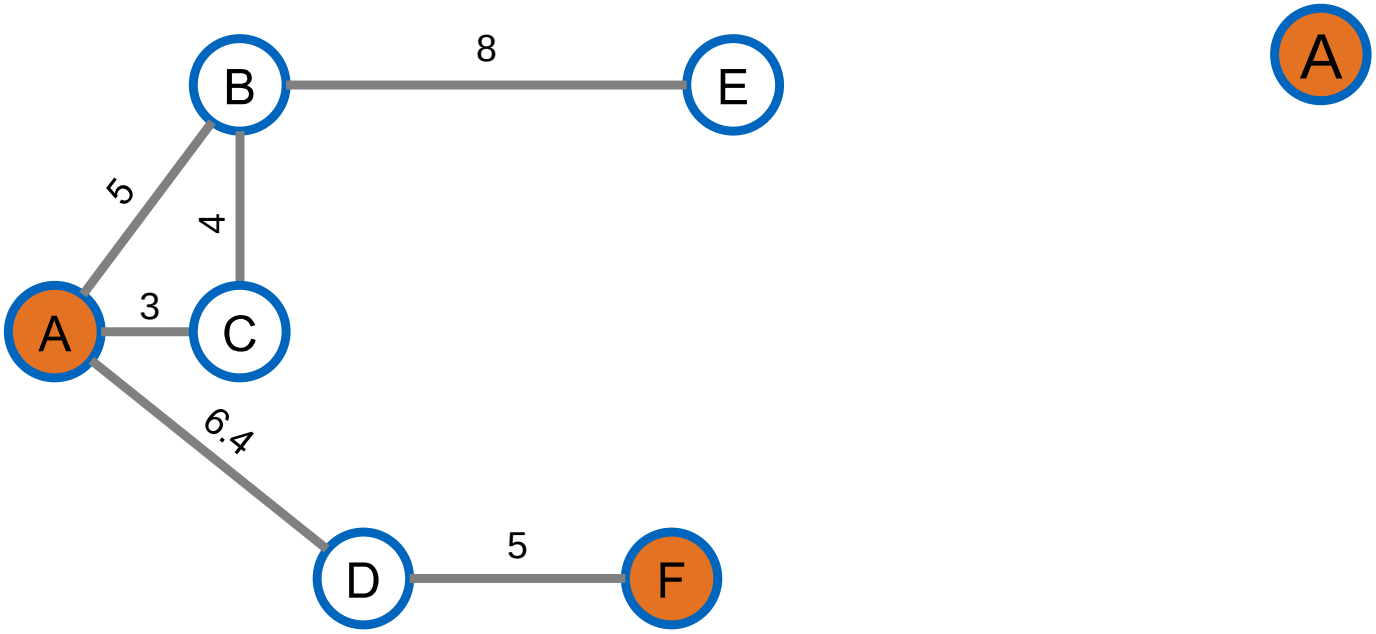
Algorithm – Core Idea:

- Until the first path in the queue terminates at the goal node or the queue is empty,
  - Remove the first path from the queue, create new paths by extending the first path to all the neighbors of the last node of the path.
  - Reject all paths with loops.
  - If two or more paths reach a common node, delete all those paths except the one that reaches the common node with the minimum cost.
  - Add the new paths, if any, to the queue
  - Sort all paths by their accumulated costs

Explore all Shortest Paths until Goal is Reached

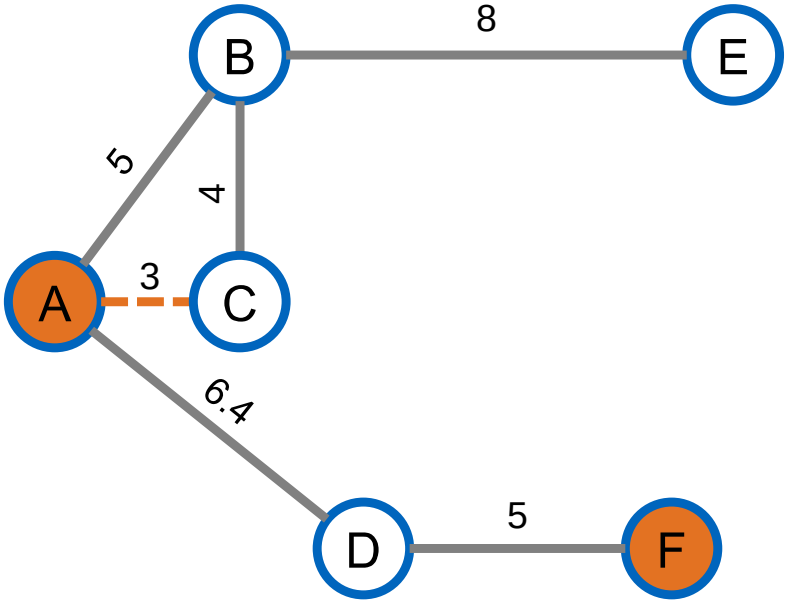
Pathfinding – Dijkstra

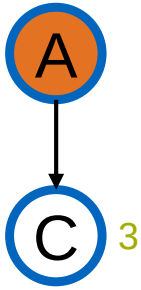
Pathfinding Example Graph



Pathfinding – Dijkstra

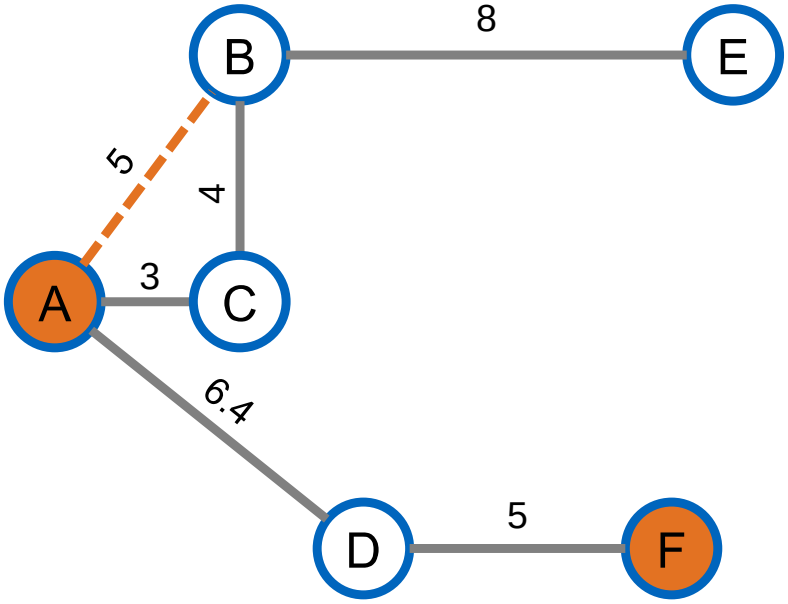
Pathfinding Example Graph

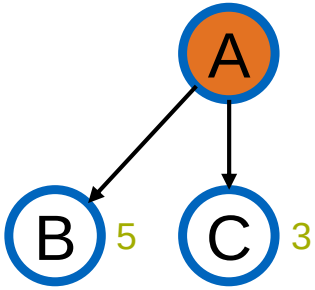




Pathfinding – Dijkstra

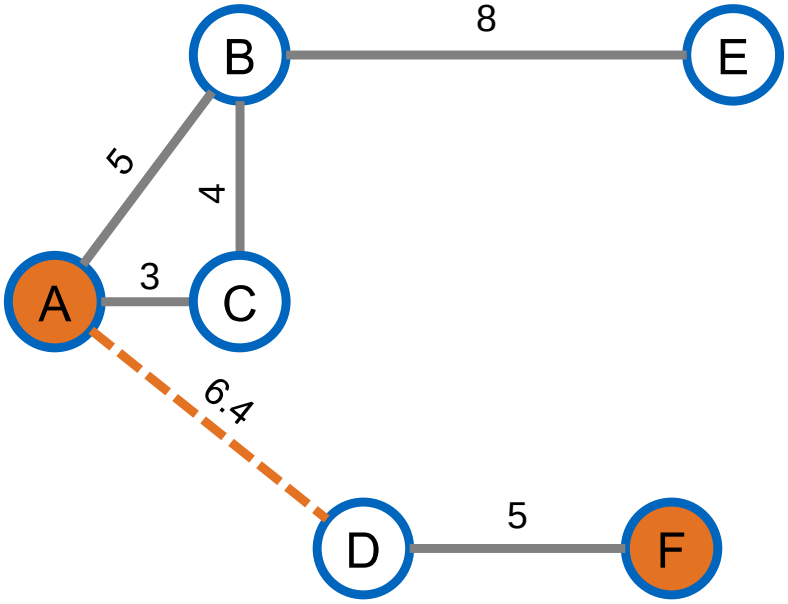
Pathfinding Example Graph

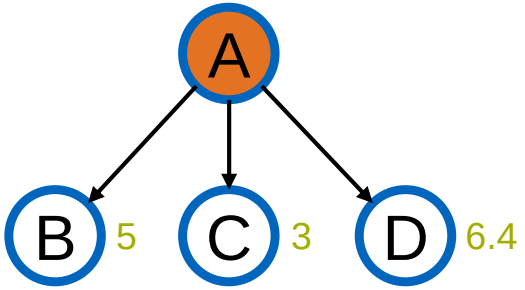




Pathfinding – Dijkstra

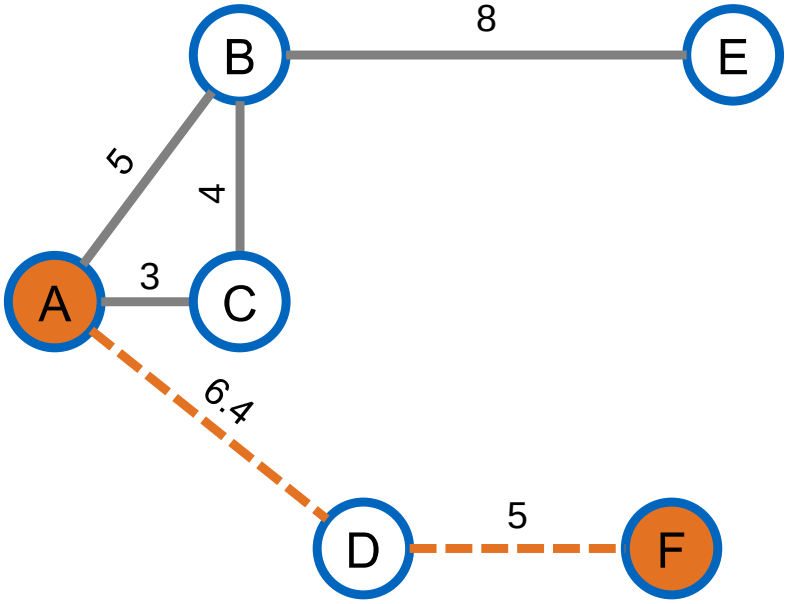
Pathfinding Example Graph

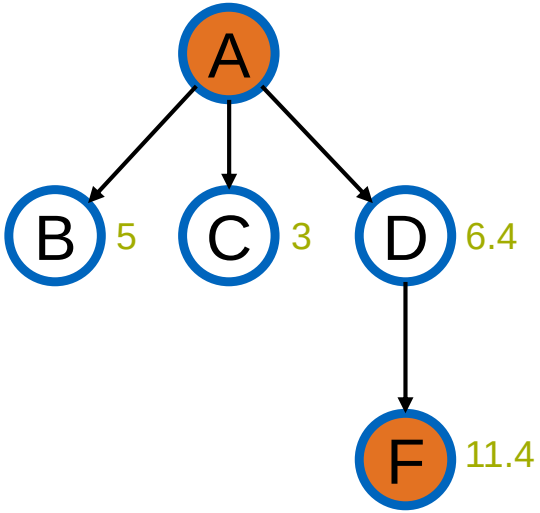




Pathfinding – Dijkstra

Pathfinding Example Graph





Knowledge Graphs: From Data to Wisdom
Prof. Dr. Markus Lienkamp

(Fabian Netzler, M. Sc.)

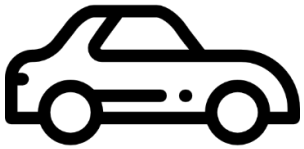
Agenda

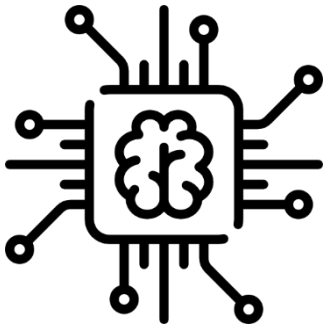
1. Chapter: Introduction into Graphs

2. Chapter: Path Planning
    - 2.1 Depth/Breadth Search
    - 2.2 Dijkstra

3. Chapter: Learning over Property Graphs
    - 3.1 Node Embedding
    - 3.2 Graph Neural Networks

4. Chapter: Summary



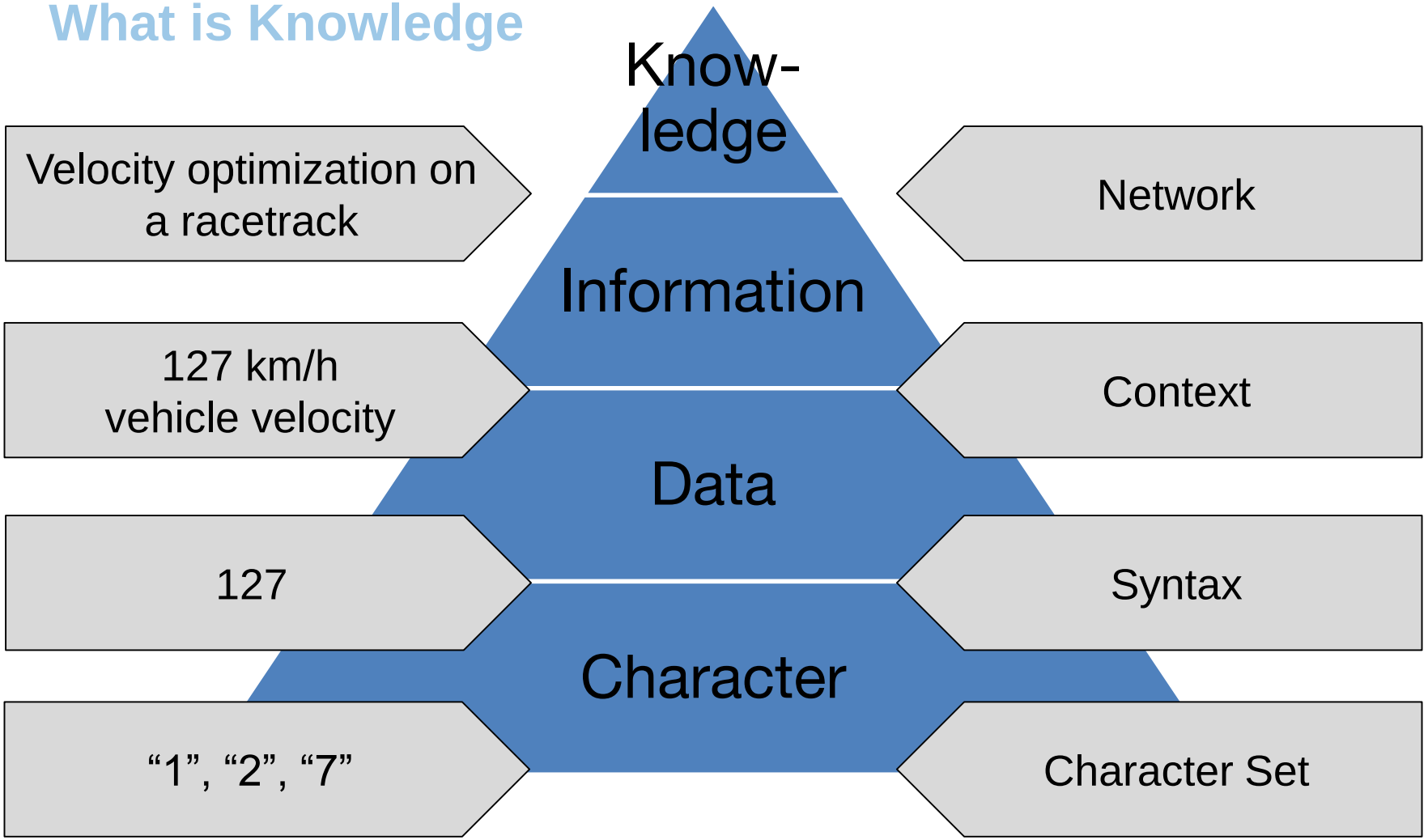


Graphs as General Purpose Data Structure

- Graphs can be used to store and organize data with strong relationships between data objects
- The restriction, that nodes have fixed coordinates is no longer valid
- Graphs can be used to link data together and in combination with specific algorithms can be referred as knowledgegraphs

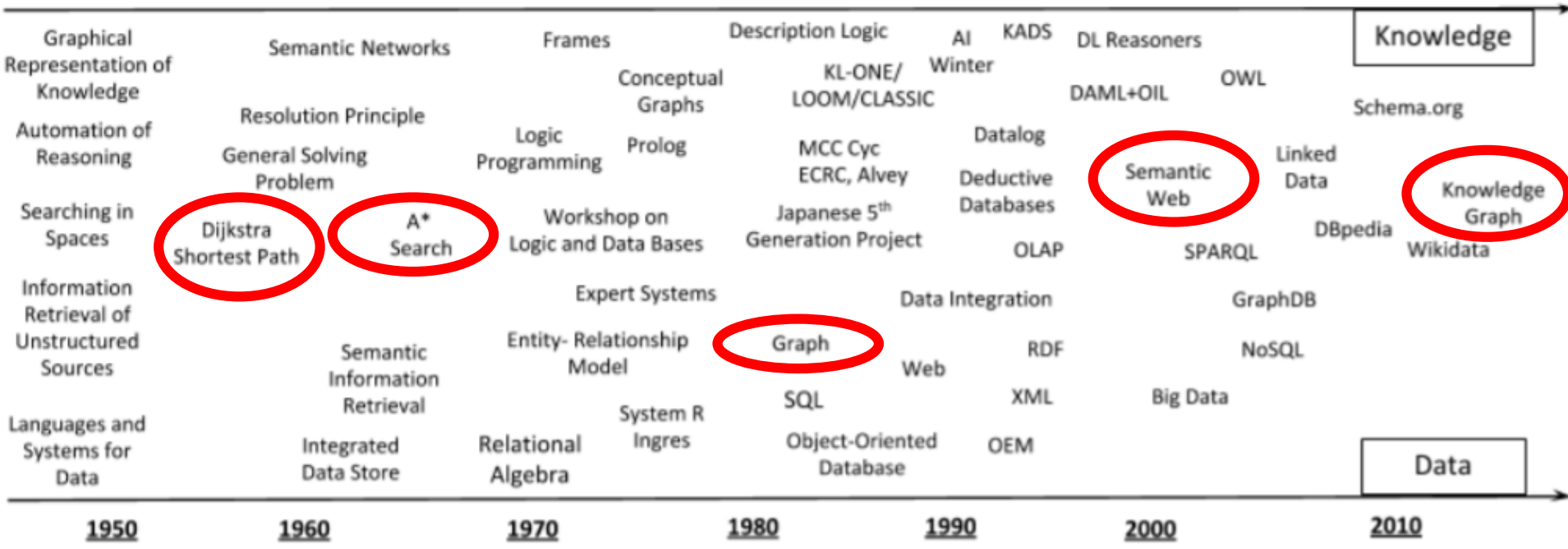
General Purpose Graphs

What is Knowledge



General Purpose Graphs

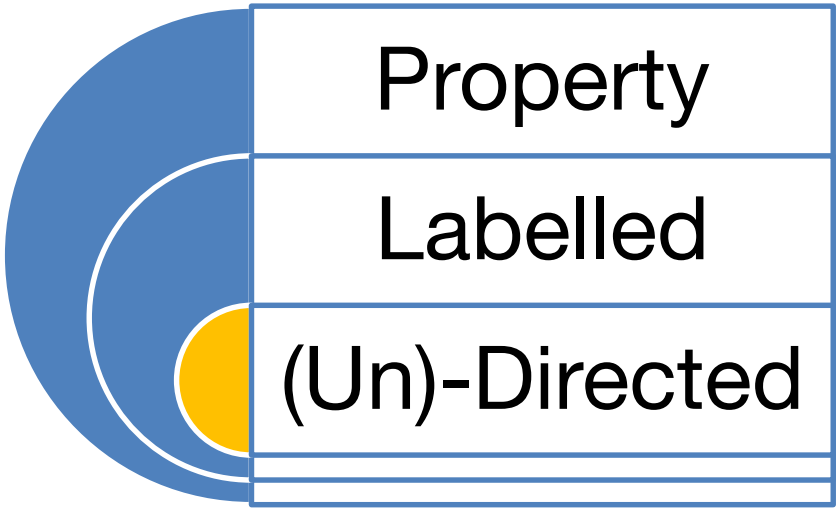
Historical Development

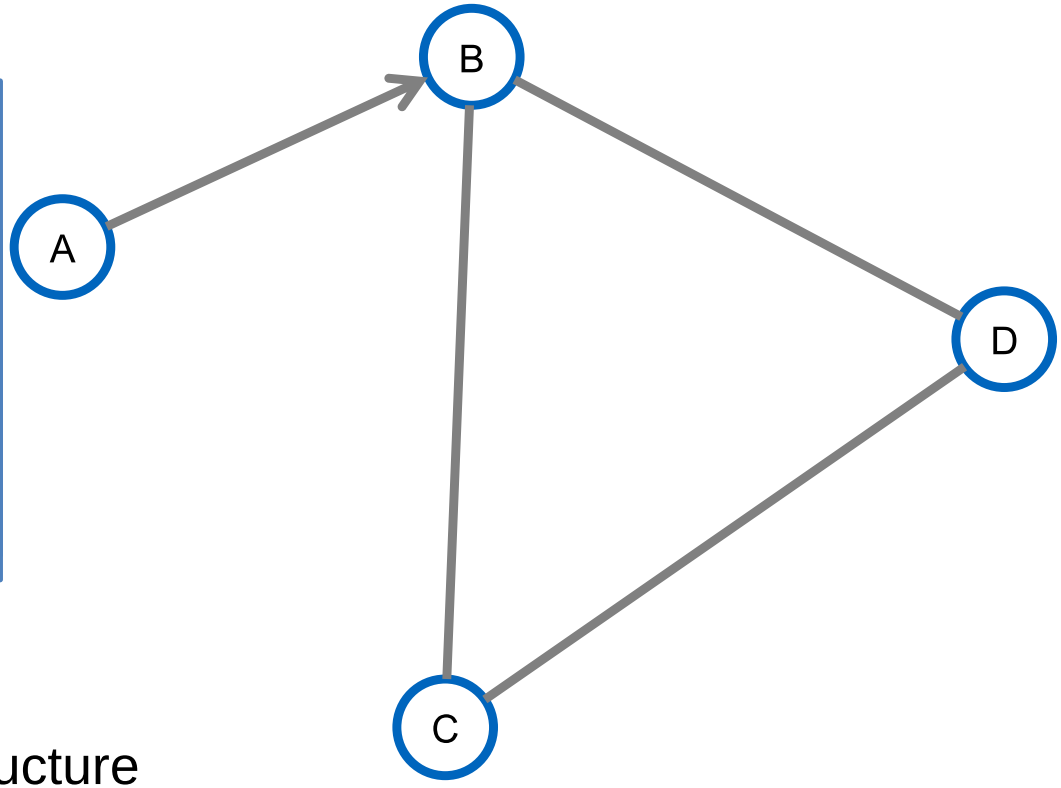


C. Guterres (2020)

Extensions to the Graph Data Model

The Base Model

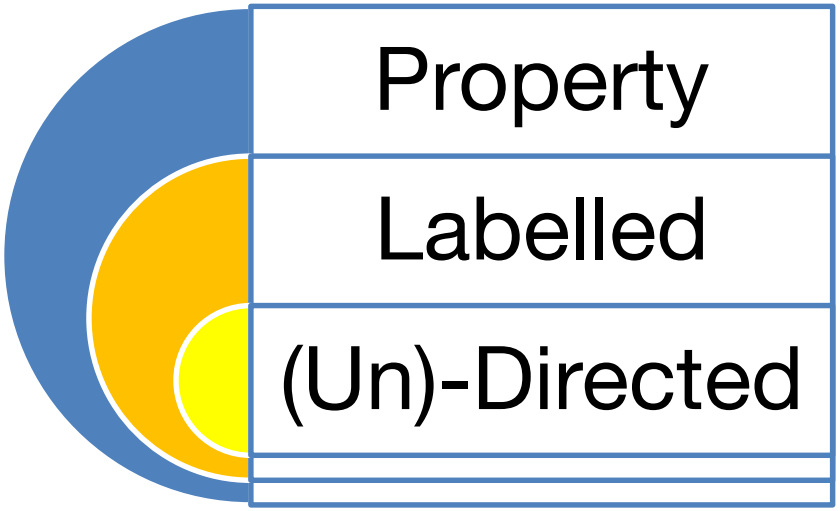




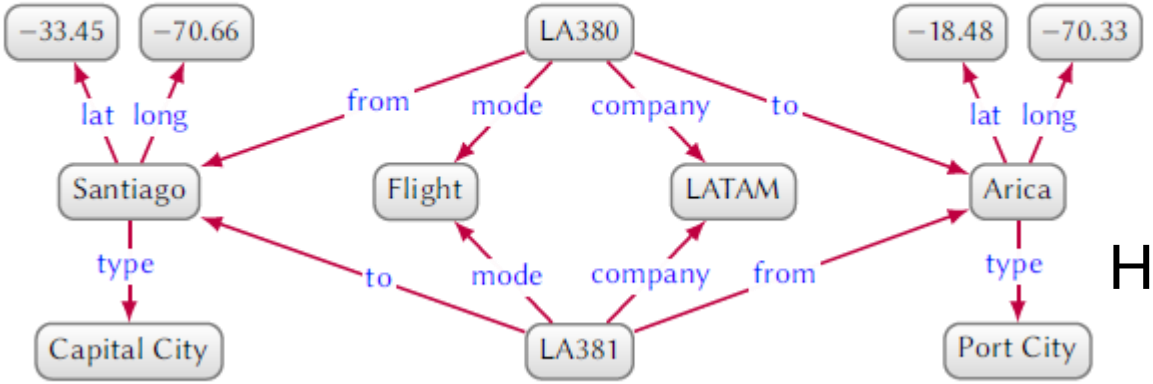
Graphs to represent a logical structure
Can be extended by the use of edge weights

Extensions to the Graph Data Model

Adding Labels



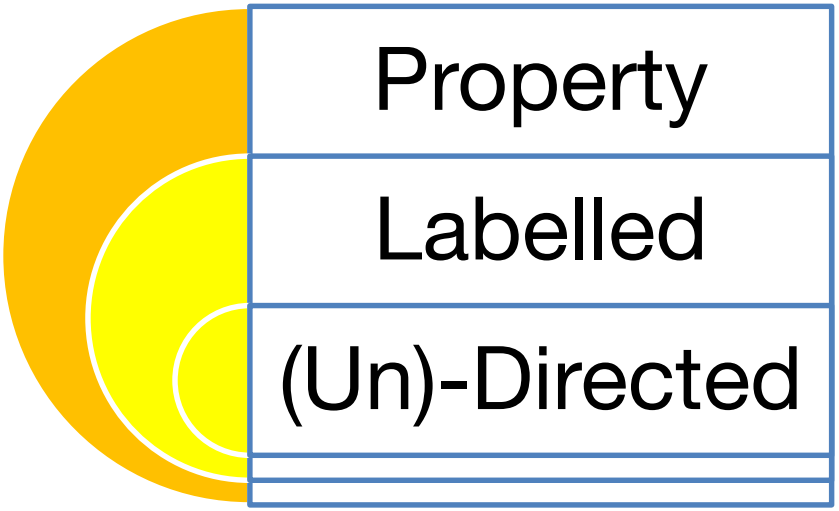
Edges are further described with labels, allowing the logical connection between different types of data objects



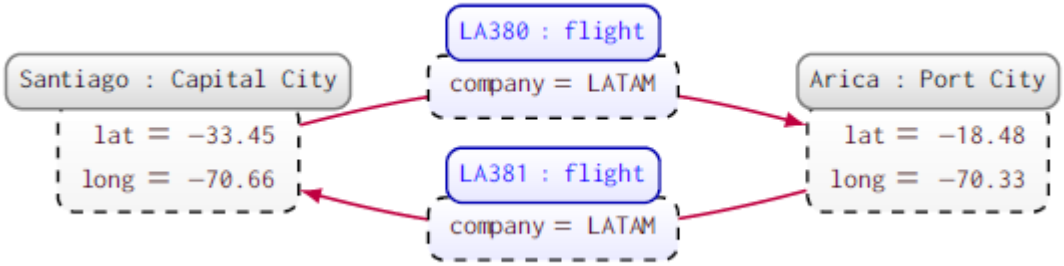
Hogan et. Al., 2020

Extensions to the Graph Data Model

Full Property Graphs



Nodes and edges are represented as complex objects with types and properties



Hogan et. Al., 2020

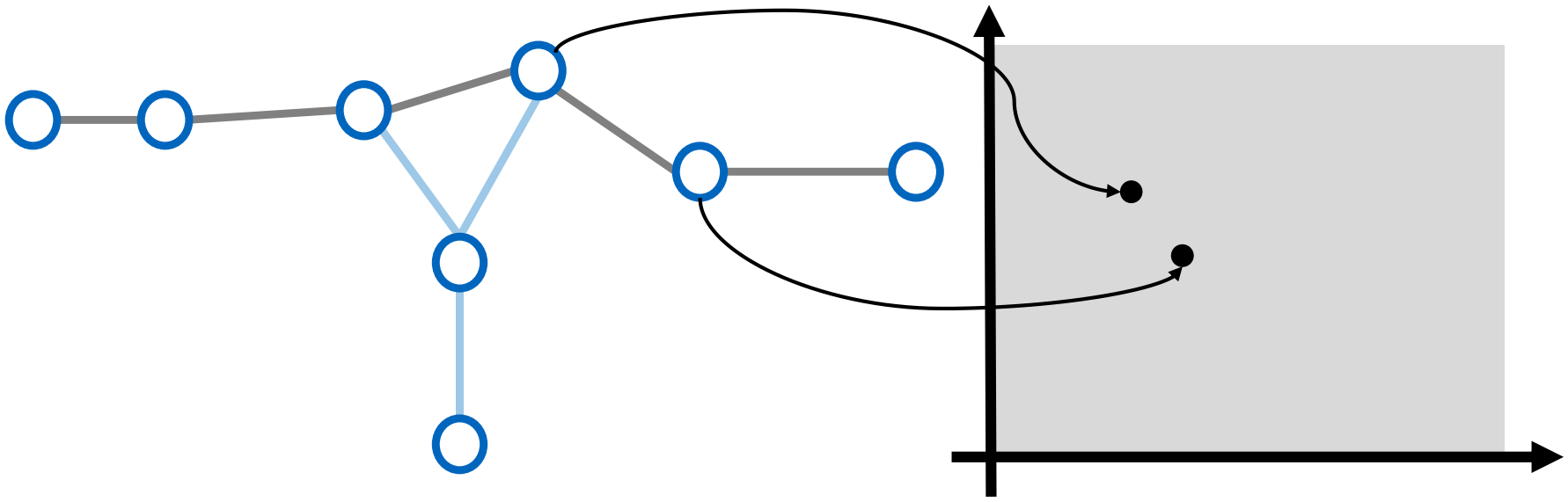
Graph Embedding

Overview

Map nodes to lower dimensional space

Find an embedding so that “similar” nodes in the graph have a close distance in the embedding.

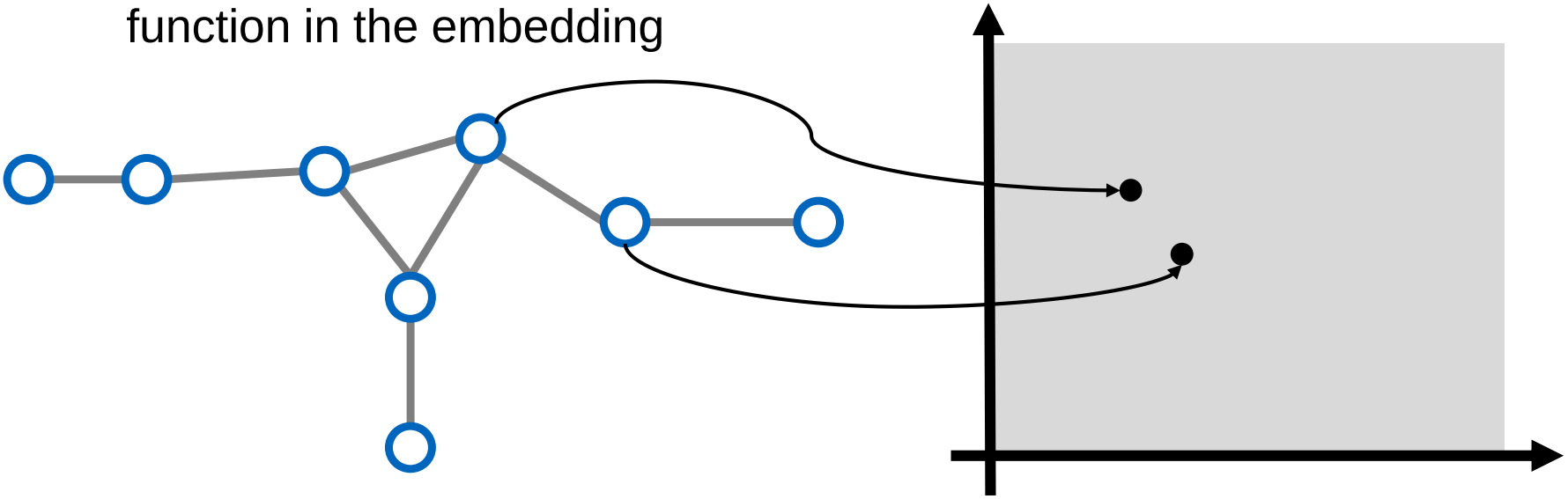
Usage of the graph structure to define similarity.



Graph Embedding

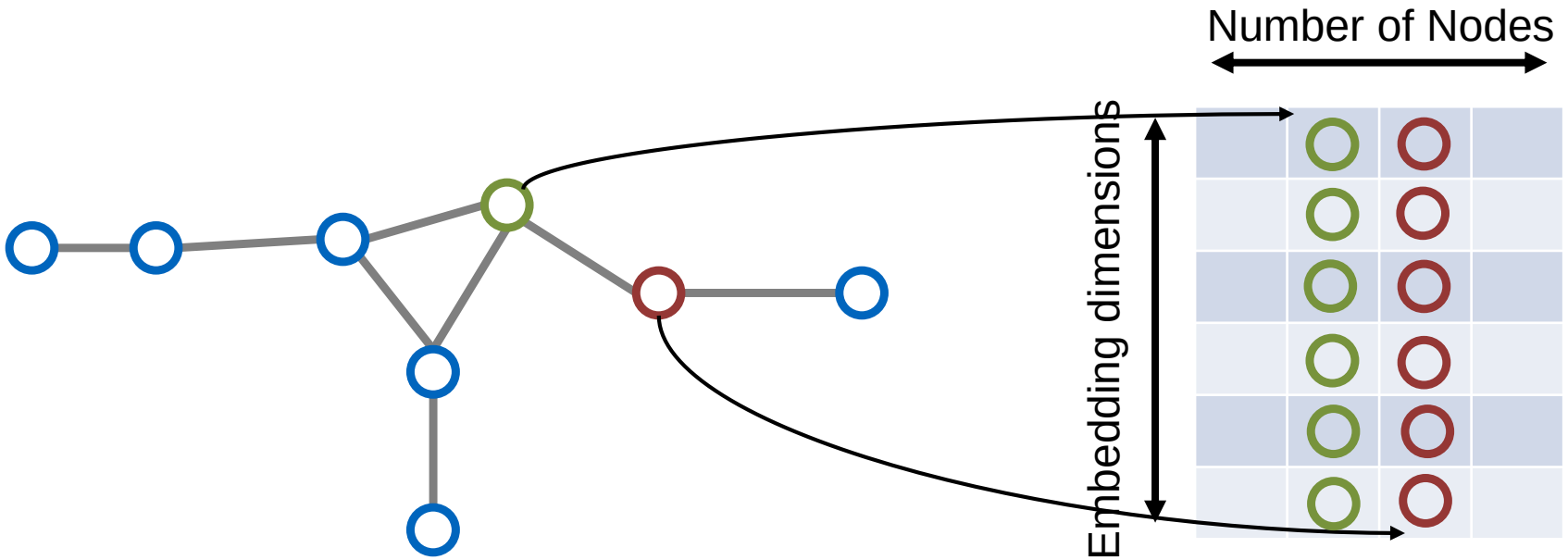
Approach

1. Find a suitable encoding from node to embedding space
$$enc(u) = z_u$$
2. Define a similarity function on the original network
$$similarity(u, v) = z_v z_u$$
3. Optimize the parameters of the encoder to reflect the similarity function in the embedding



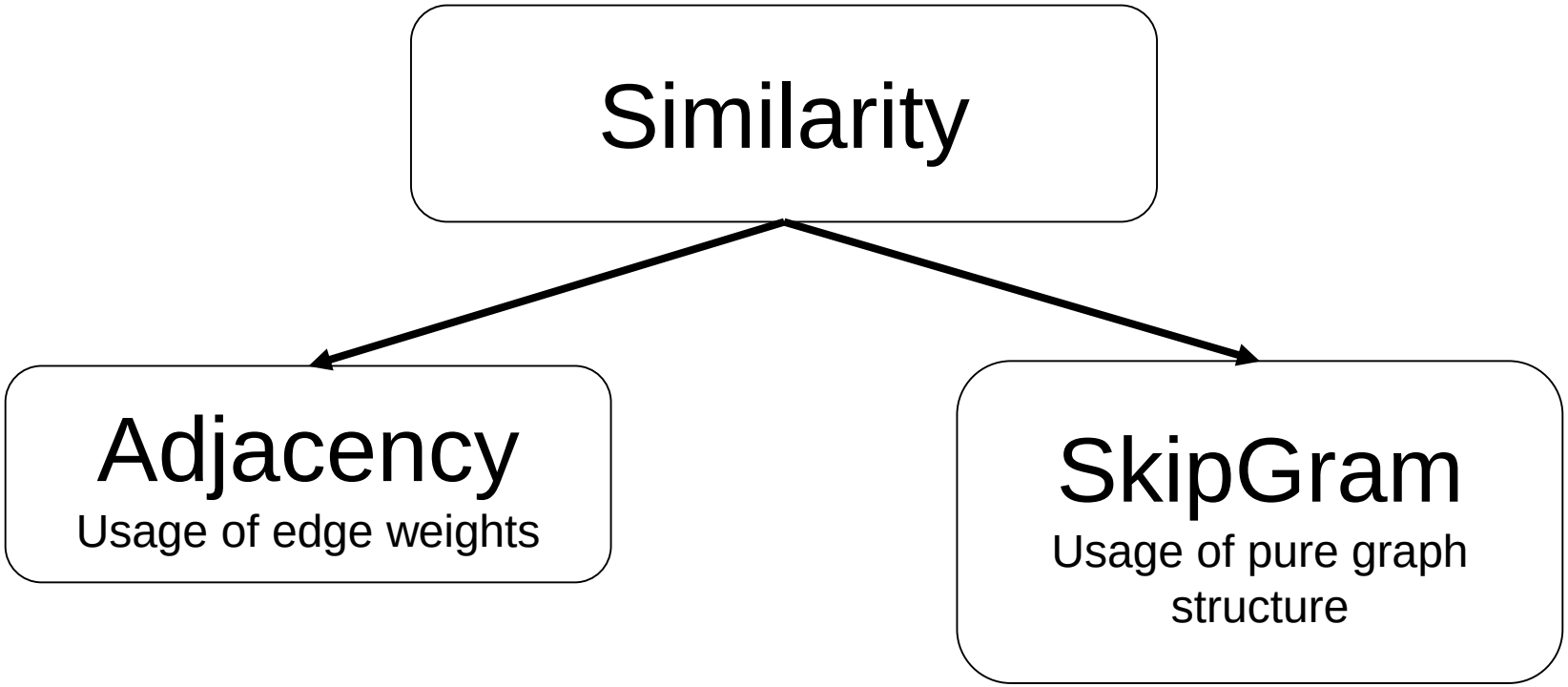
Graph Embedding
Shallow Embedding

Shallow Embedding The embedding function is a lookup table with a column for each node and the number of rows representing the dimensions of the embedding space



Shallow Embedding

Defining Similarity

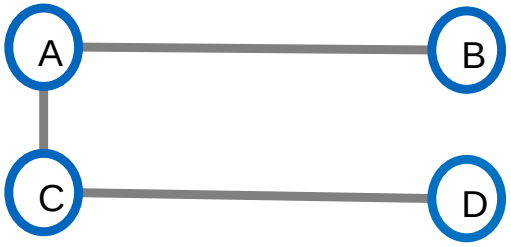


Shallow Emebdding

Adjacency based Similarity

Adjacency matrix *A* nxn-matrix with the neighborhood information of each node.

Nodes	A	B	C	D
A	0	1	1	0
B	1	0	0	0
C	1	0	0	1
D	0	0	1	0

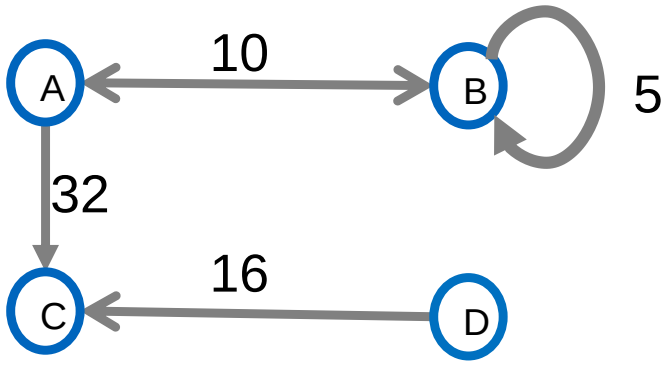


Shallow Emebdding

Adjacency based Similarity

Adjacency Matrix *A*:

Nodes	A	B	C	D
A	0	10	32	0
B	10	5	0	0
C	0	0	0	0
D	0	0	16	0



Shallow Emebdding

Adjacency based Similarity

Similarity: Use the edge weight (or reversed edge weight) between the nodes as similarity metric.

Loss Function:

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \left\| z_u z_v \; - A_{u,v} \right\|^2$$

Shallow Emebdding

Adjacency based Similarity

Goal Find embedding matrix A that minimizes the loss function.

Loss Function

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \left\| z_u z_v - A_{u,v} \right\|^2$$

Method Numerical optimization, like (stochastic) gradient descent

Drawbacks

Runtime and number of parameters

Considers only direct connections

Shallow Embedding

SkipGram - Approach

Similarity Definition Chance of the nodes *u* and *v* being both present on a random walk over the graph.

Approach

1. Use a random walk algorithm to generate a set of node sequences

2. Estimate the probability of visiting node u when using a random walk strategy starting from node *v.*

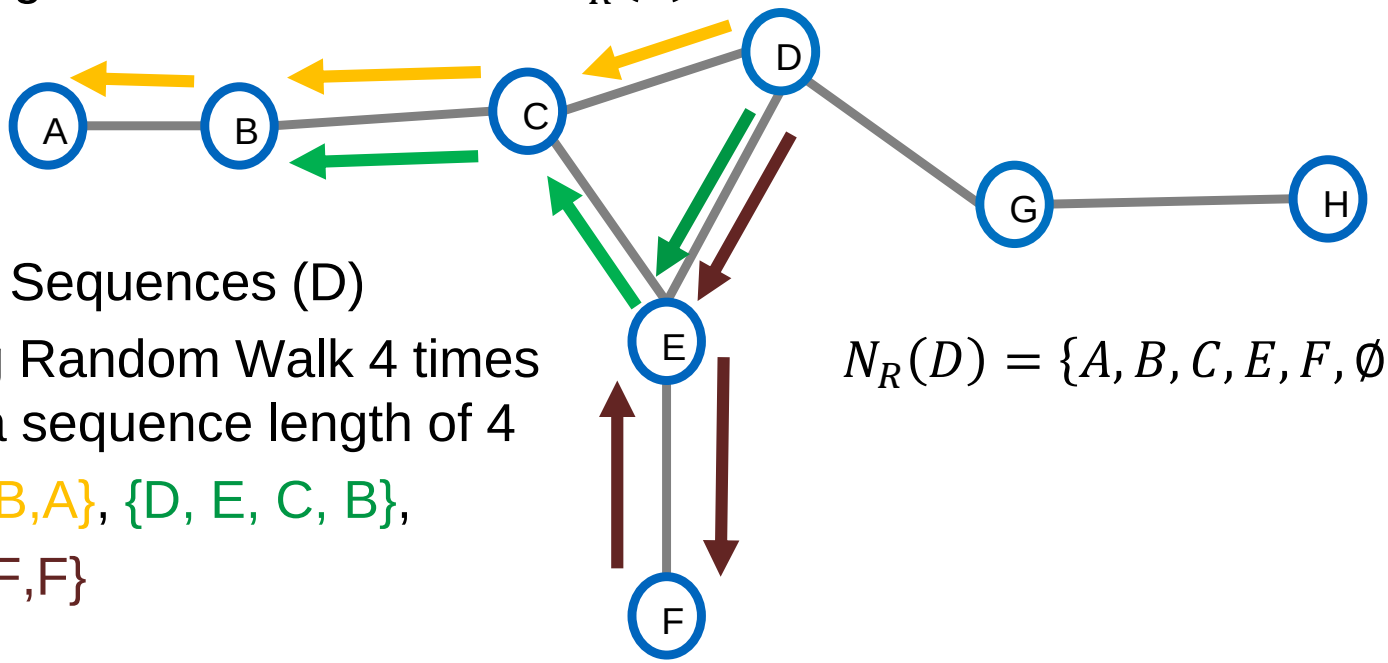
3. Optimize the embedding to encode the generated statistic from step 1.

Shallow Embedding

Random Walk – Step 1 – Generate Sequences

Gather a set of node sequences with fixed length from the graph.

For each node *u* gather the multiset of nodes visit on random walks, starting from *u*, described as $N_R(u)$.



Node Sequences (D)

Using Random Walk 4 times with a sequence length of 4

{D,C,B,A}, {D, E, C, B},

{D,E,F,F}

$N_R(D) = \{A, B, C, E, F, \emptyset\}$

Shallow Embedding

Random Walk – Step 2 – Loss Function

Using the Multiset $N_R(u)$ a loss function can be defined:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(z_u z_v)}{\sum_{n \in V} \exp(z_u z_n)}\right)$$

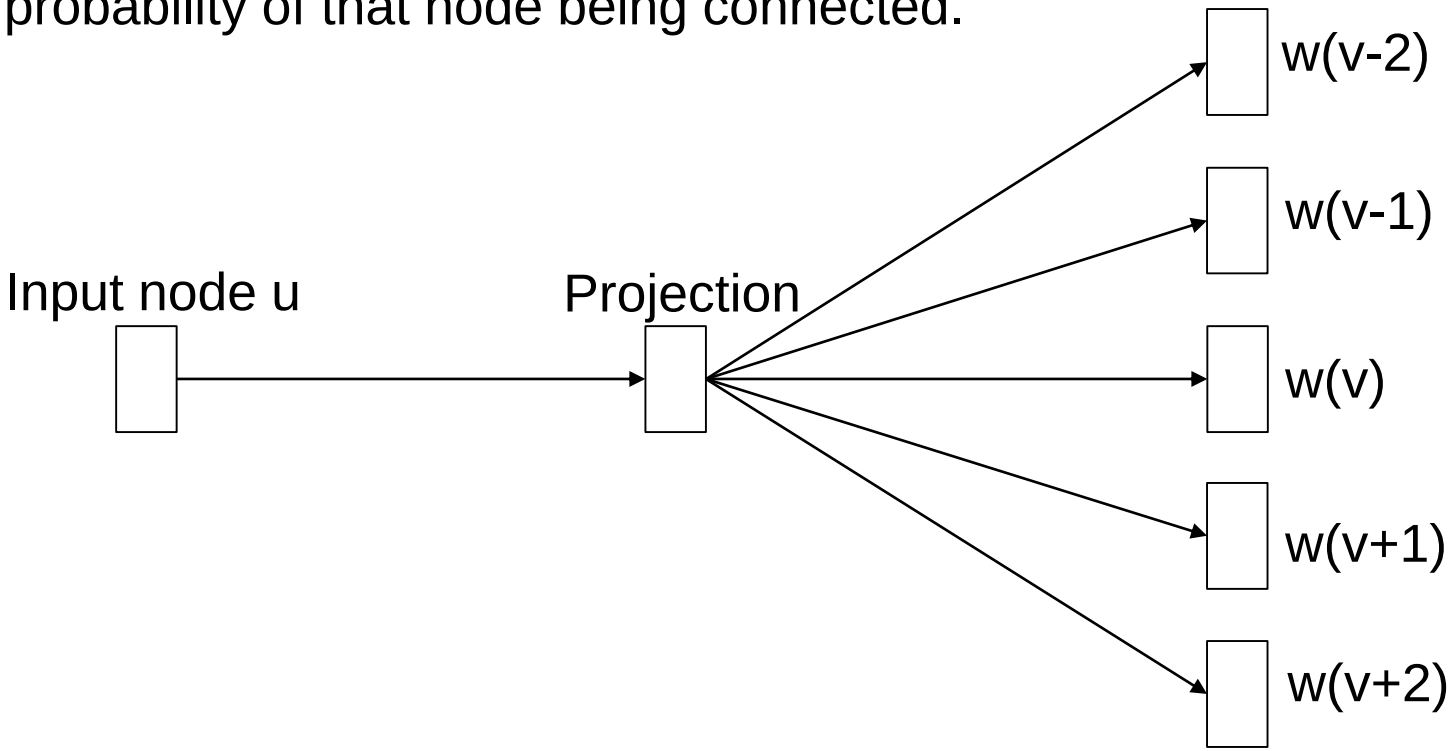
Every node in the graph

Every node in the sequence superset

Shallow Embedding

Random Walk – SkipGram

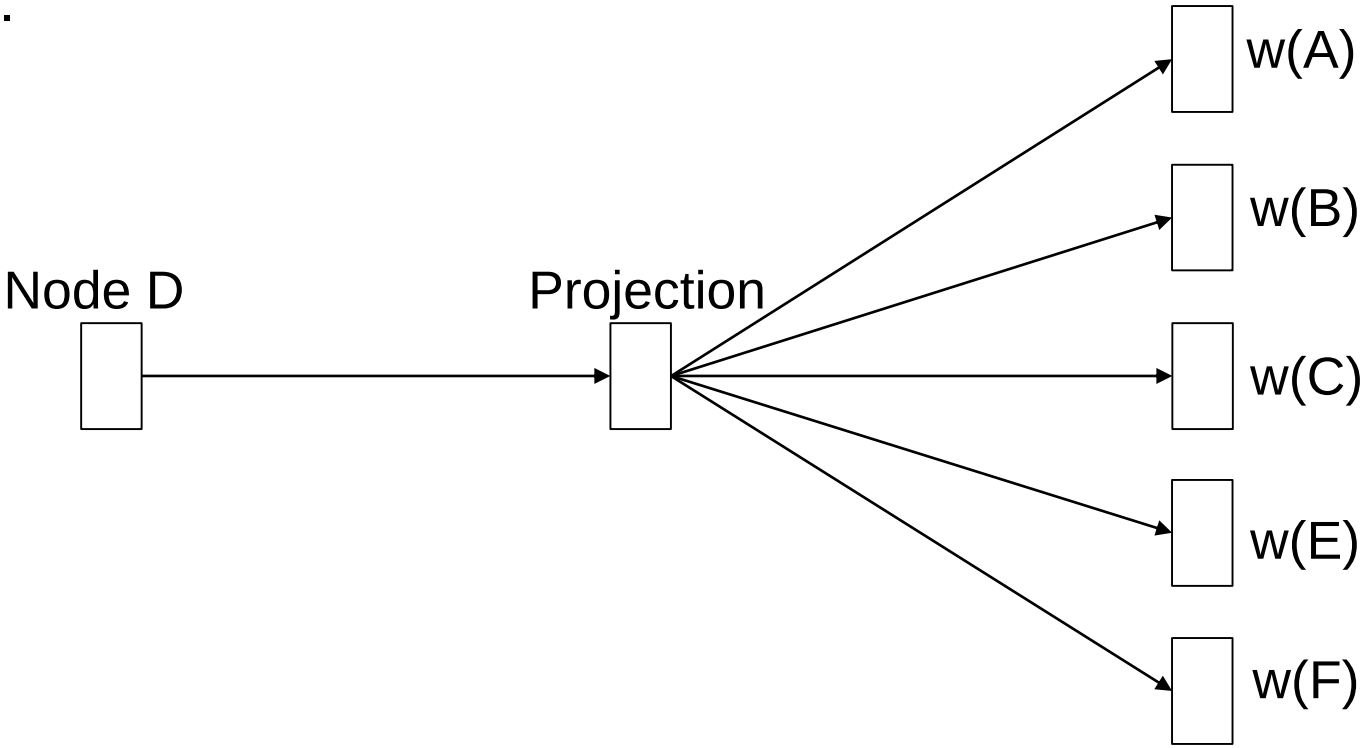
Idea Train a simple neural network with one hidden layer for each node. Given that node, pick another random node, the network gives the probability of that node being connected.



Shallow Embedding

Random Walk – SkipGram

Solution The network is trained with the random walk sequences. The weights of the hidden layer form the embedding of the input node.



Shallow Embedding

Random Walk – SkipGram- Architecture

Input: A Vector of the length |N|, all zeroes, except the position of the node, currently looked at. On this position a one is set.

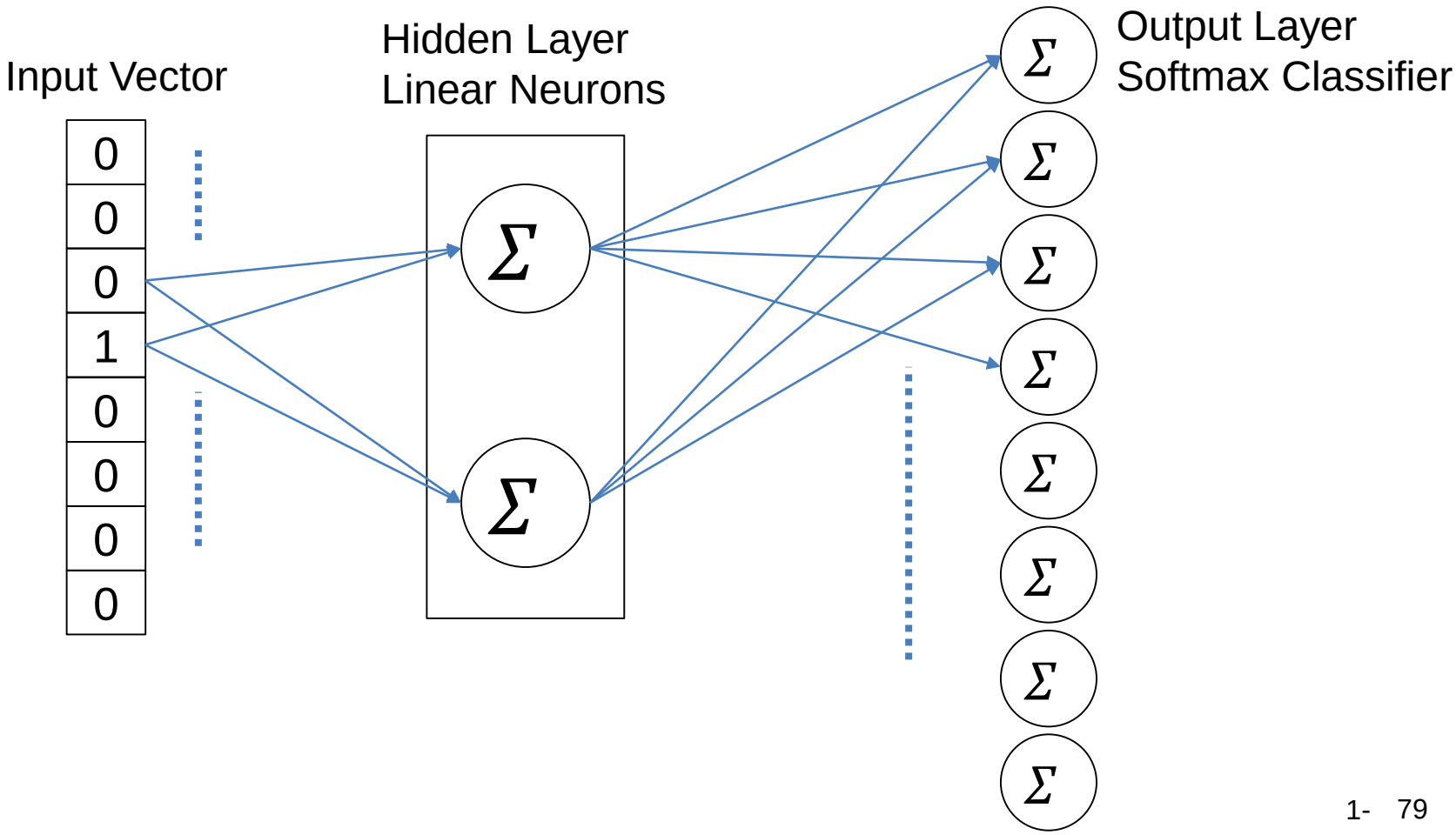
Hidden Layer: Number of neurons are equal to the dimensions in the embedding space, no activation function is given.

Output Layer Probability of the node z, that it appears with the input node in a random walk sequence

Shallow Embedding

Random Walk – SkipGram- Architecture

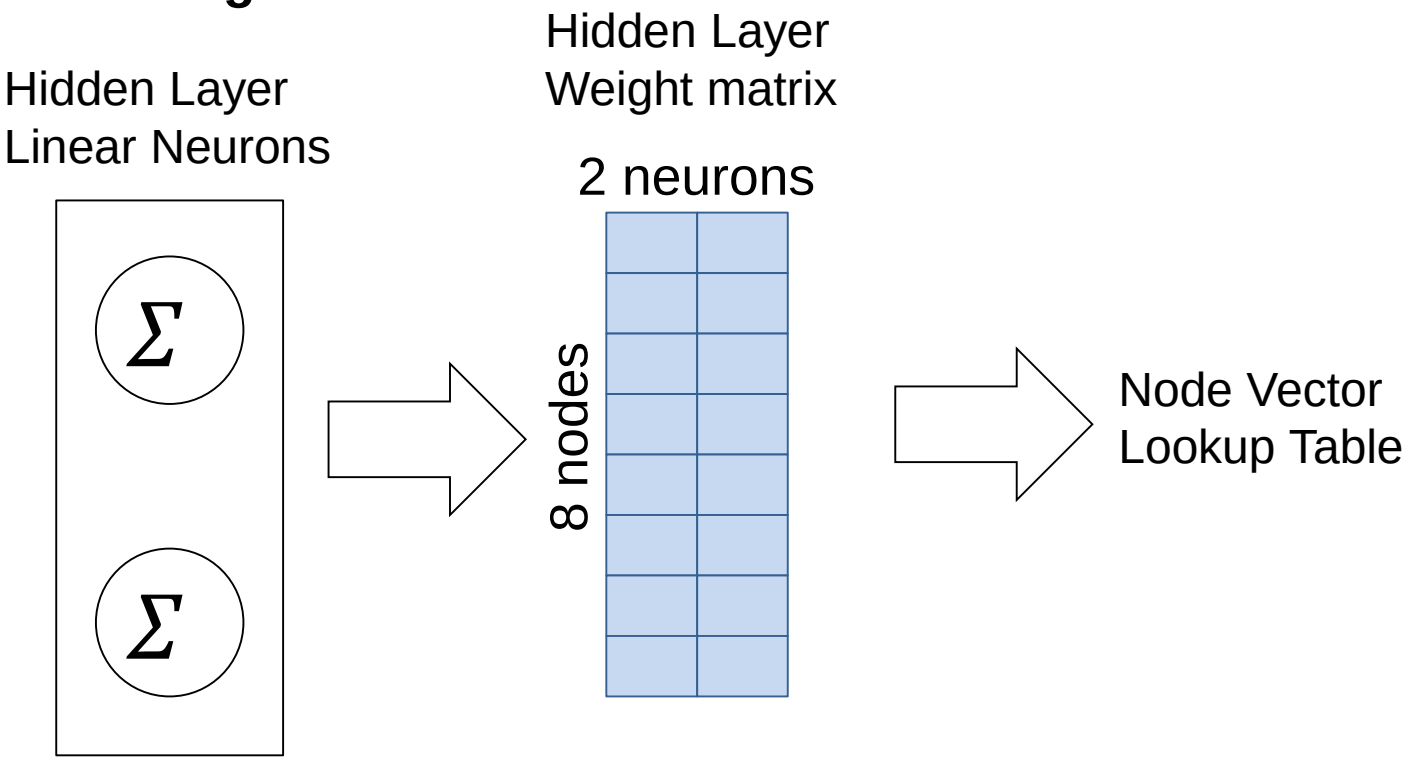
Embedding



Shallow Embedding

Random Walk – SkipGram- Architecture

Embedding



The Graph Neural Network Model

A *Graph Neural Network* builds a neural network based in the topology of the data graph (Hogan et. Al., 2020)

Features are represented as properties of nodes and edges.

Symbol	Description
$G = (N, E)$	Graph G, consisting of N nodes and E set of edges
$ne(n)$	Set of neighbors of node n
$co(n)$	Set of Edges connected to n
$l_n, l_{(n1,n2)}$	Label/Properties of node n or the edge n1-n2

The Graph Neural Network Model

General Approach

The graph is used as model architecture.

The Model is trained on the graph.

The output is represented in the graph as additional property per node.

The Graph Neural Network Model

General Approach

Core Idea Propagate the node information over the network to combine data features with the logical structures and data relationships.

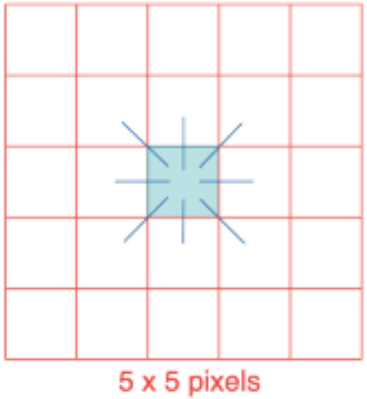
In Addition to the label/property vector each node gets a hidden state vector ***h*** assigned

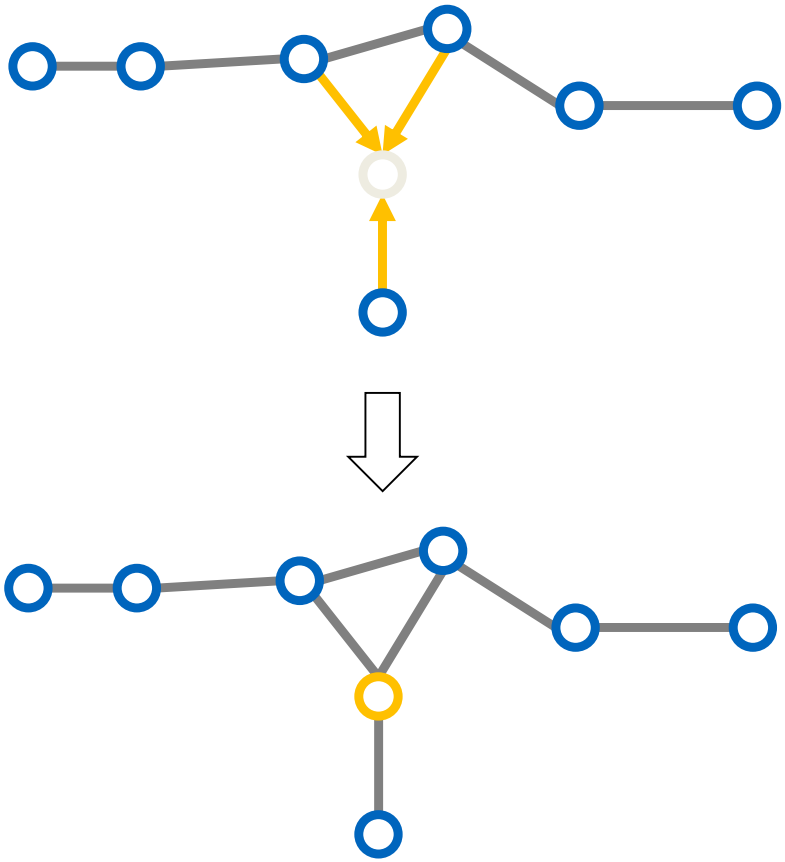
h is dependent on the time step *t.*

h can be initialized with the feature vector for *t=0.*

Graph Convolutional Neural Networks (GCN)

Approach





Graph Convolutional Neural Networks (GCN)

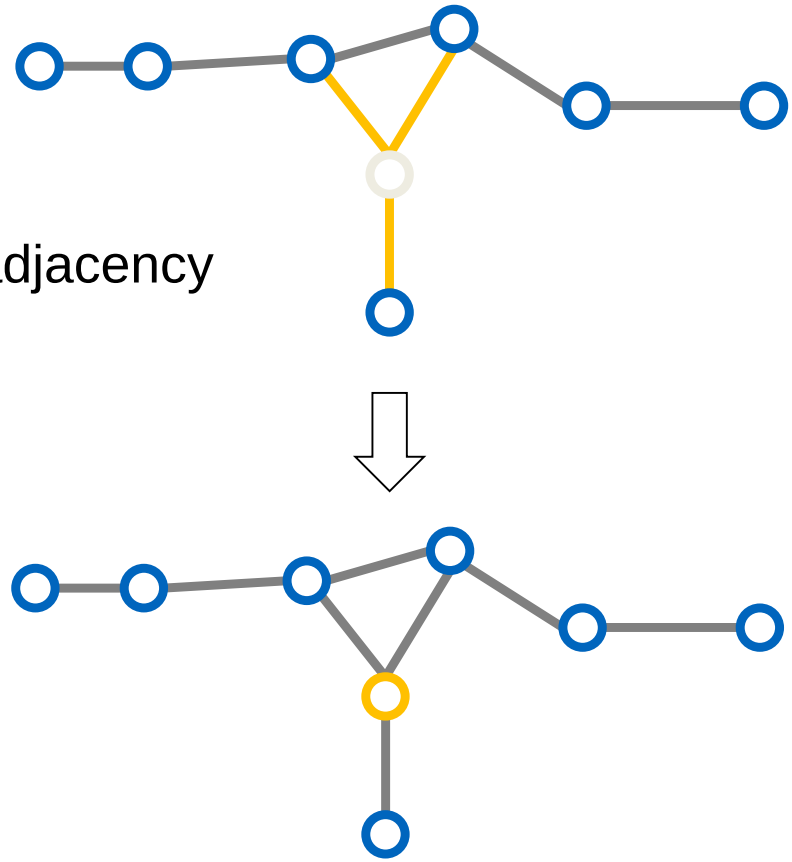
Mathematical Definition

Use the adjacency matrix *A*

Aggregate the neighborhoods by multiplying the features with the adjacency matrix and learnable weights *W*.

Feed this to a non-linear function.

$$h^{t+1} = \sigma(Ah^tW)$$



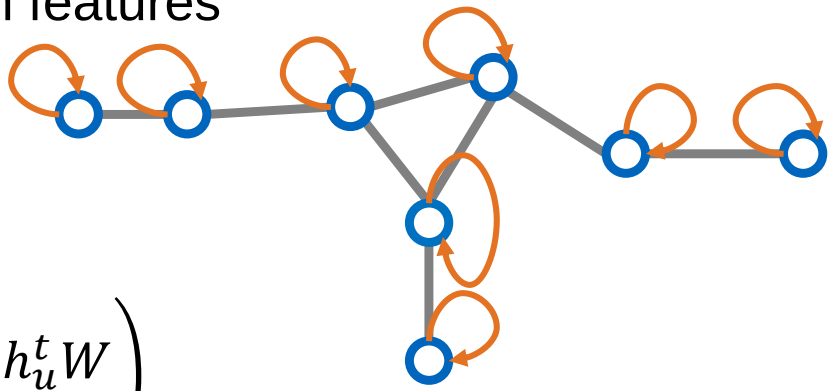
Graph Convolutional Neural Networks (GCN)

Mathematical Definition

2 Aspects are missing

1. The node should also use its own features

$$\tilde{A} = A + I$$



2. The output should be normalized

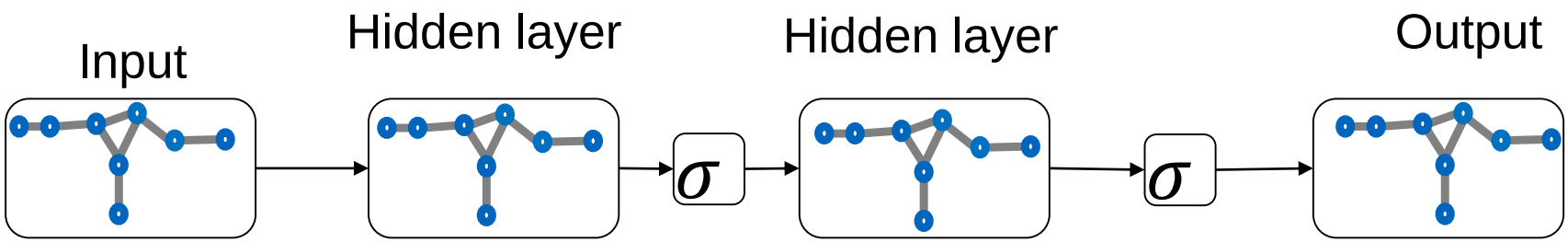
$$h_v^{t+1} = \sigma\left(\sum_{u \in N(v)} \frac{1}{|N(v)|} h_u^t W\right)$$

In most literature the symmetric normalization is chosen

$$h_v^{t+1} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{|N(v)||N(u)|}} h_u^t W\right)$$

Graph Neural Network

Multi-Layer Networks

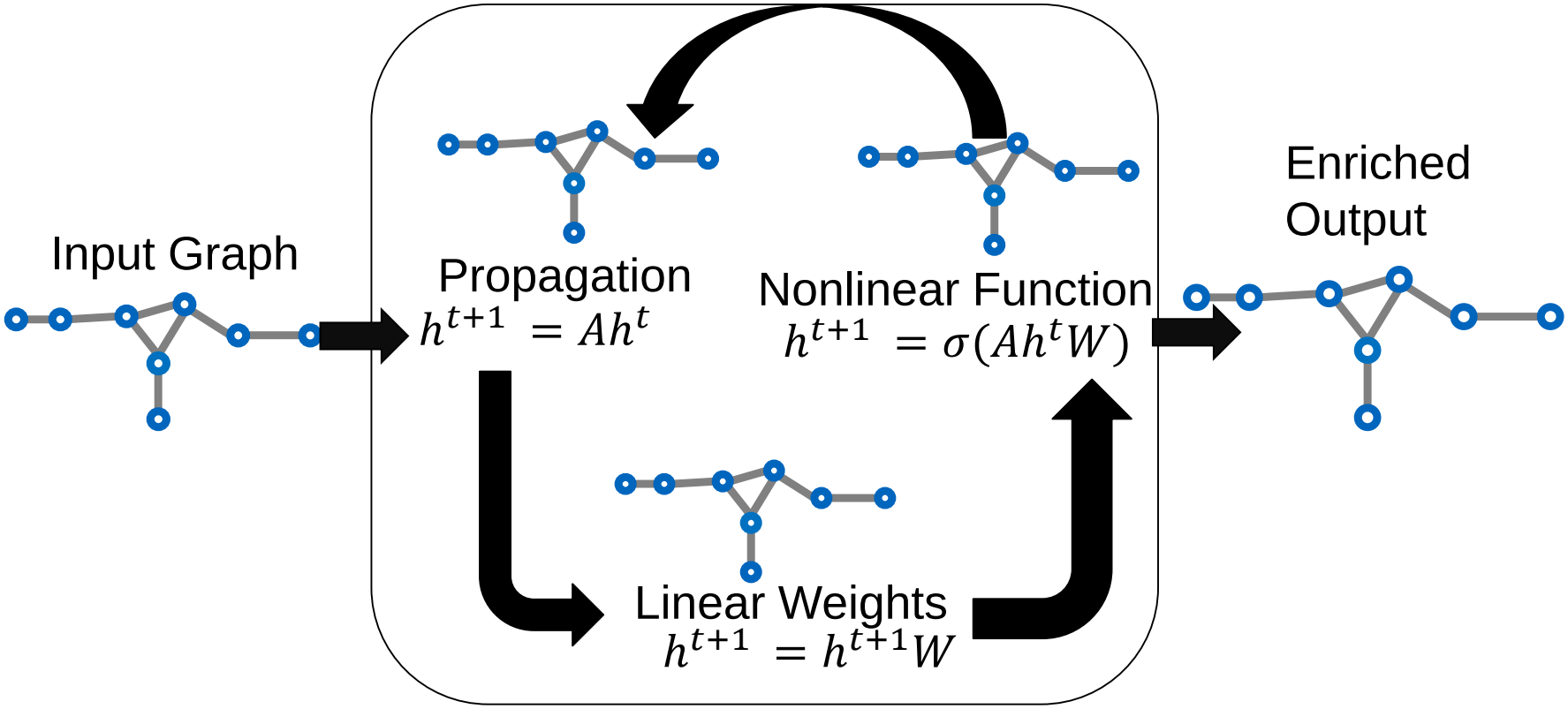


Training through backpropagation, like a conventional CNN.

Like a CNN, the training is done through kernels,
defined as the message passing function.

Graph Convolutional Neural Networks (GCN)

Mathematical Definition



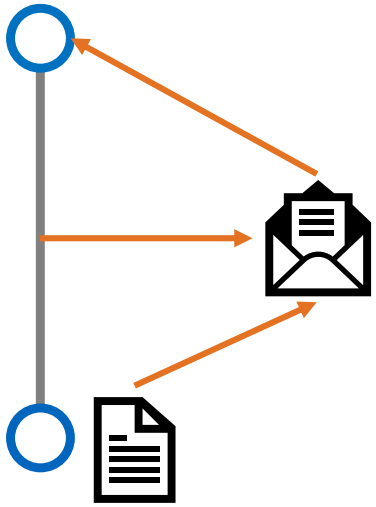
Graph Convolutional Neural Networks (GCN)

From GCN to Message Passing

Adjacency based Algorithms do not support edge features.

Instead edge-focused algorithms can be used

Idea Nodes send messages along the edges, these can be conditioned by the edge features. A node aggregates all messages sent to it.



Message Passing
Hidden State Update

General formula for *h*

$$h_v^{(t)} = \sum_{u \in N(v)} f(l_v, l_{(v,u)}, l_u, h_v^{(t-1)}, h_u^{(t-1)})$$

Example for *f*

$$f = \frac{h_u^{(t-1)}}{|N(v)|}$$

The output is generated by combining the hidden state and the feature vector of a node.

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v)$$

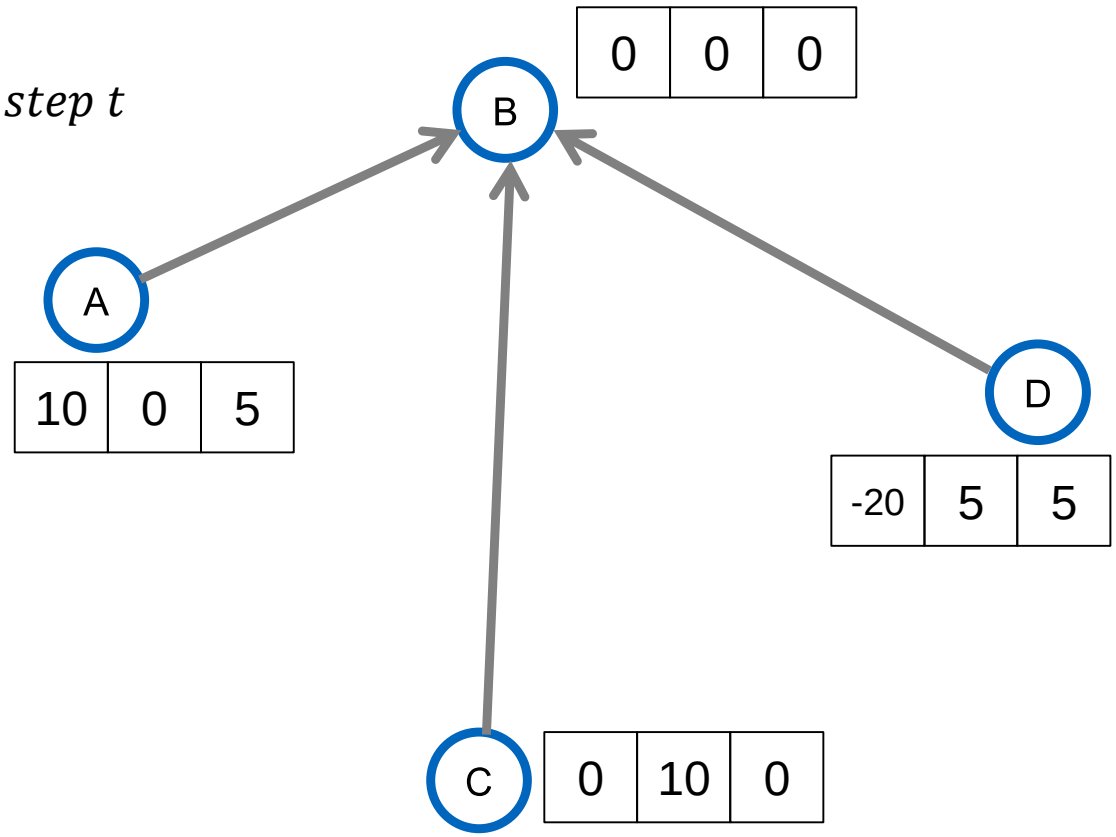
Message Passing Example

Example

h_v^t: *Hidden state for node v in step t*

$$h_v^{t+1} = avg(h_v, m_v^{t+1})$$

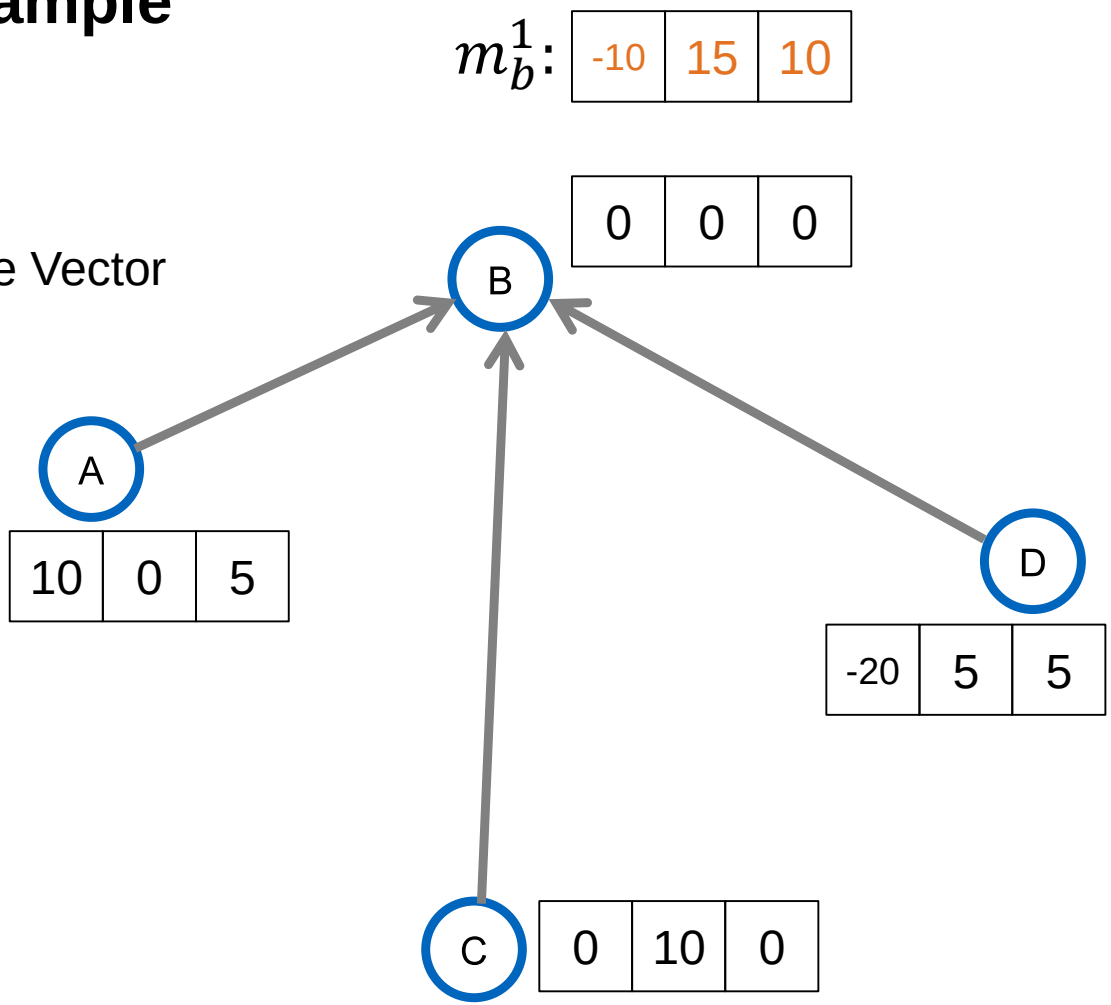
$$m_v^{t+1} = \sum_{w \in N(v)} h_w^t$$



Message Passing Example

Example

Creating the aggregated Feature Vector for all surrounding nodes

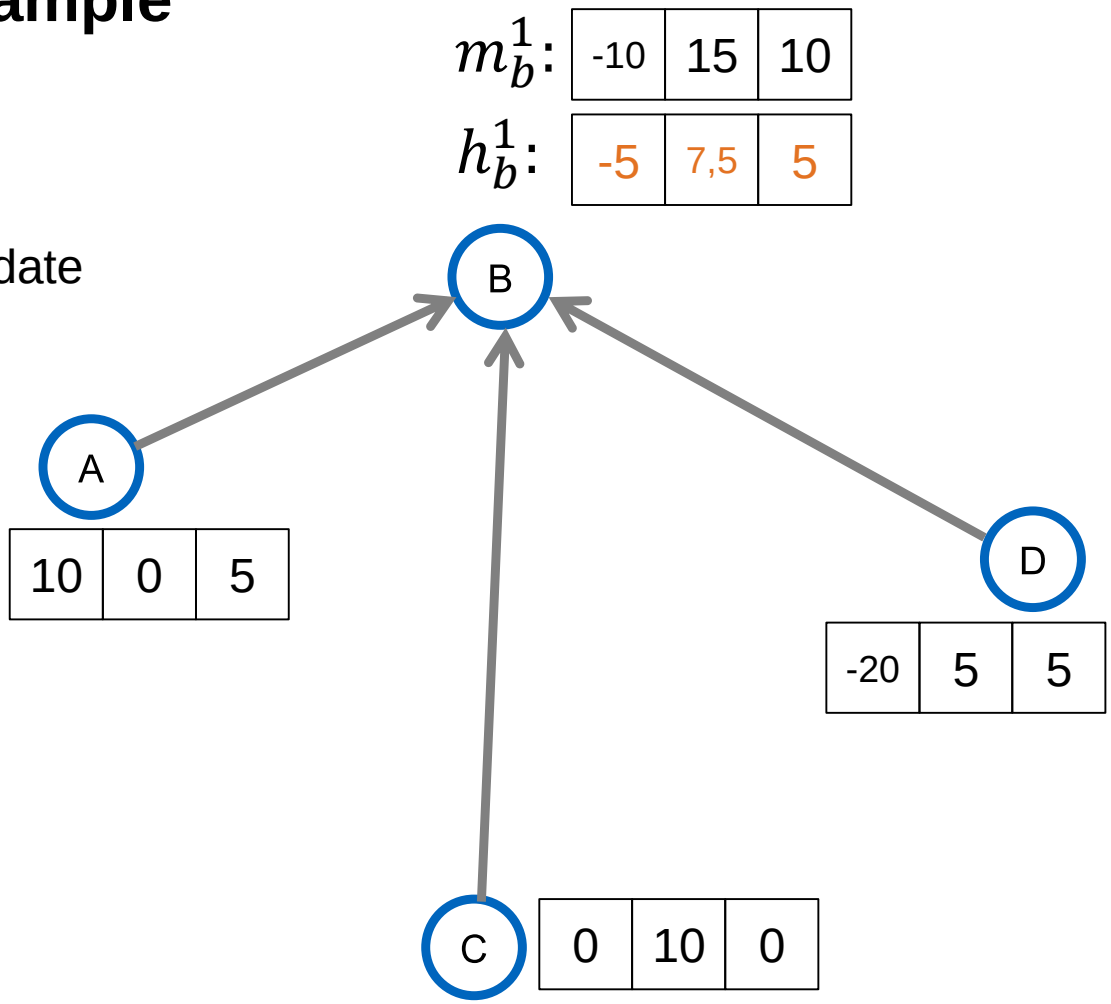


Message Passing Example

Example

One iteration of hidden state update for node B.

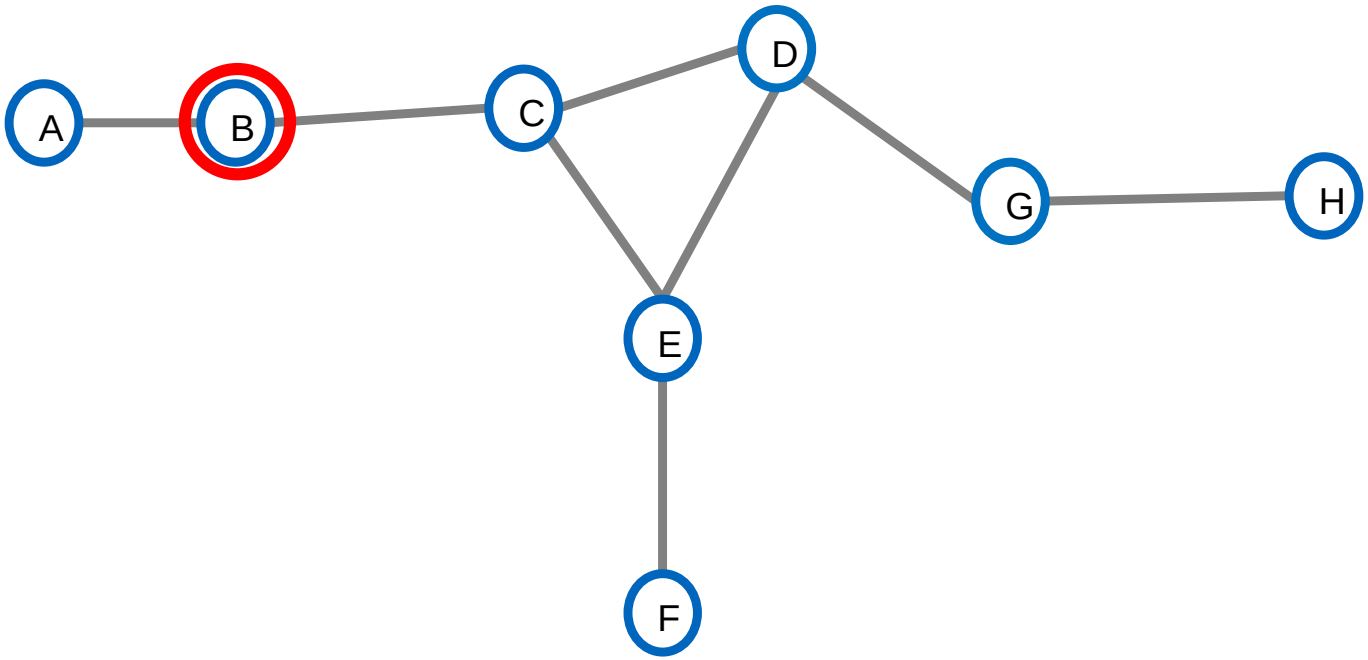
Repeat fixed number of times or until a break condition is reached



Graph Neural Network

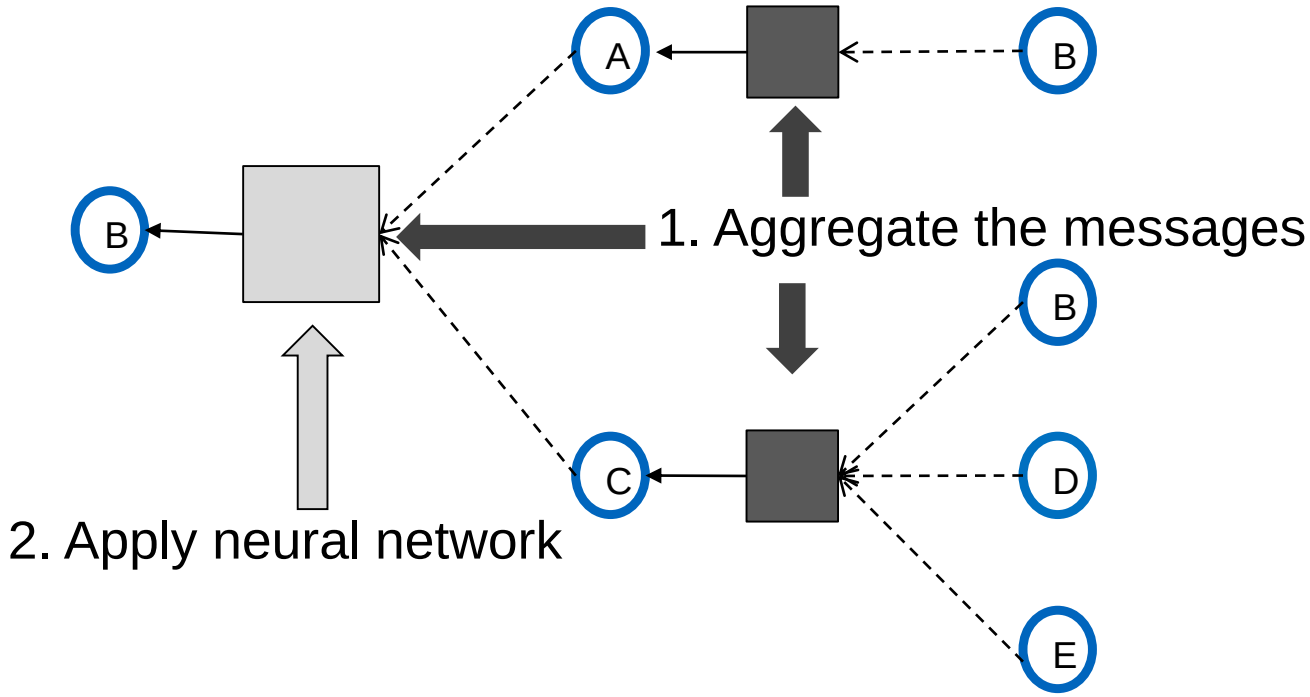
Approach

Idea Use the average neighbor information from the message passing and apply a neural network on it.



Graph Neural Network

Approach



Graph Neural Network

Training the model

$$h_v^0 = l_v$$

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

Use a non-linear function σ for the hidden state update

Use a loss function and gradient descent to train the matrices ***W*** and ***B***.

Graph Neural Network

Supervised Learning

Use data labels *y* (e.g. node classification) to model a loss function.

$$\mathcal{L} = \sum_{v \in V} y_v \log\big(\sigma(z_v \theta)\big) + (1 - y_v) \log(1 - \sigma(z_v \theta))$$

y_v: Ground truth for node v

z_v: Output of the node embedding

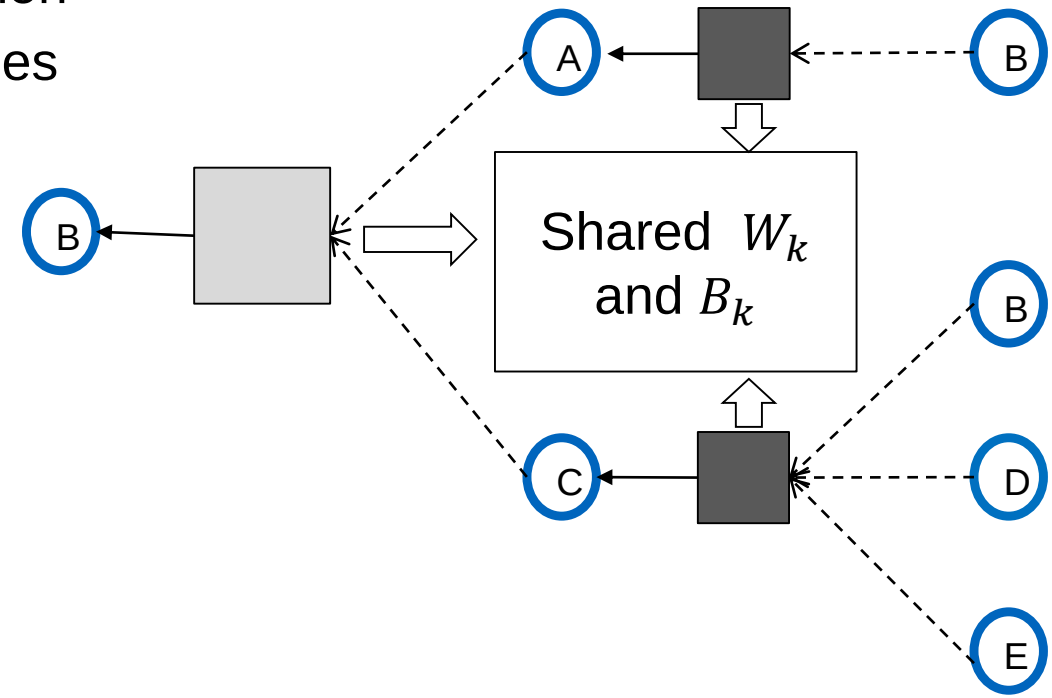
θ: classification weights

Graph Neural Network

Overview Model Design

1. Define the message aggregation function
2. Define the loss function
3. Train on a set of nodes

The same aggregation parameters are shared over all nodes.



Development Nowadays

Heterogenous Graph Neural Networks

Graph Query Language and Connection with relational data storage

Useful Python packages

Networkx

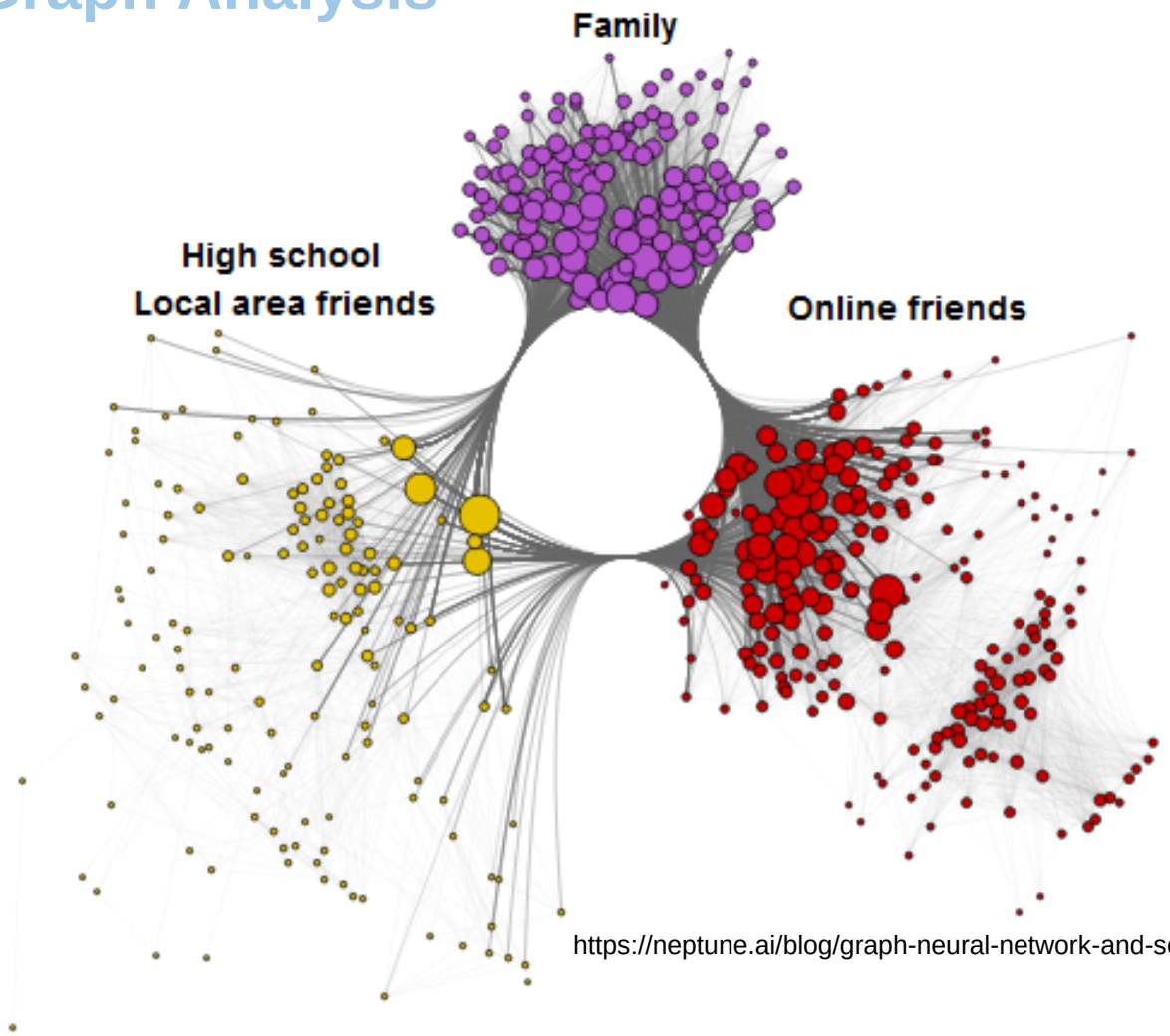
Scikit.learn.geometry

Applications

- **Node classification**
- **Link prediction**
- **Graph clustering**
- **Graph classification**
- **Computer vision**

Applications

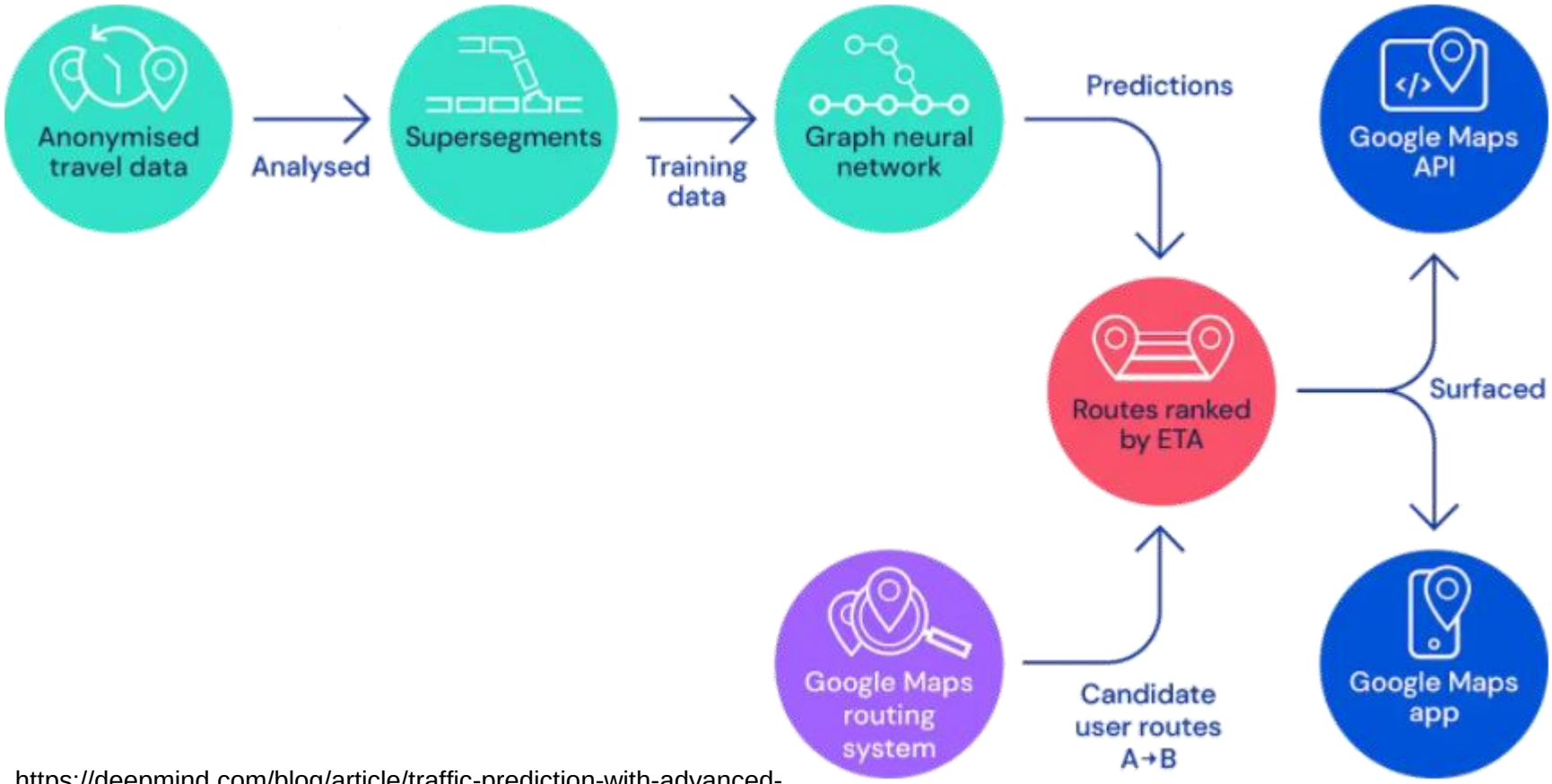
Social Graph Analysis



https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications

Applications

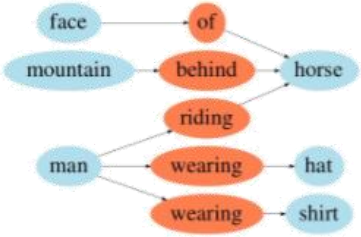
Deepmind Travel Time Estimation

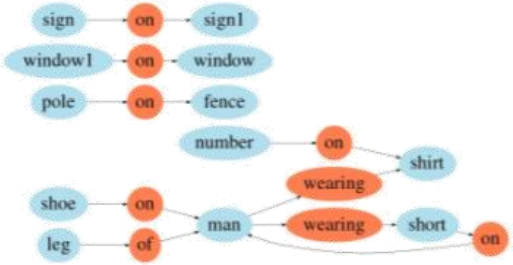


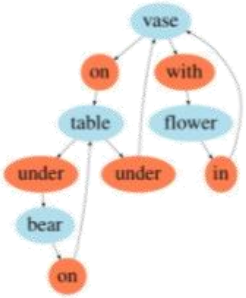
https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks

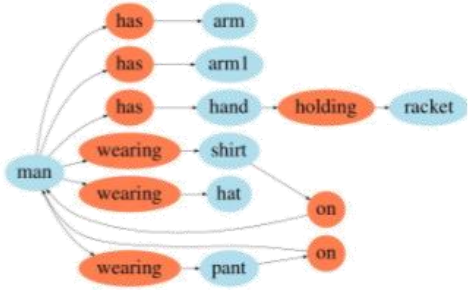
Applications

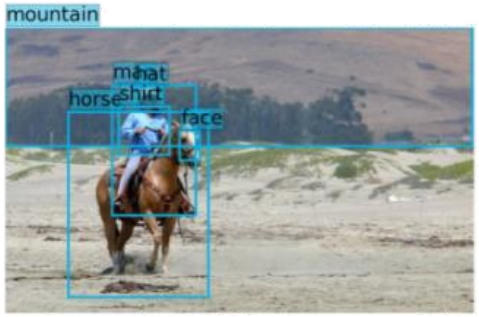
Scene Understanding

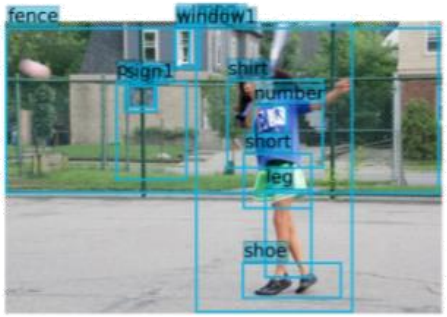


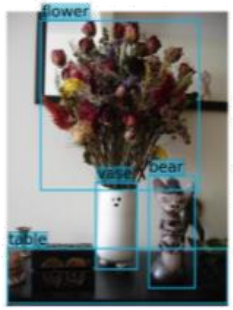


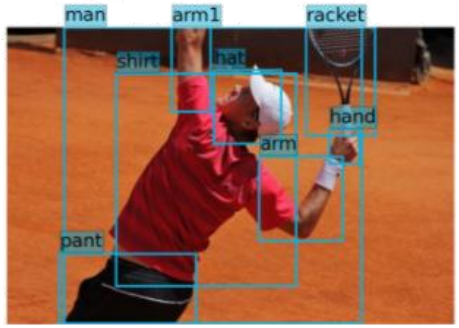




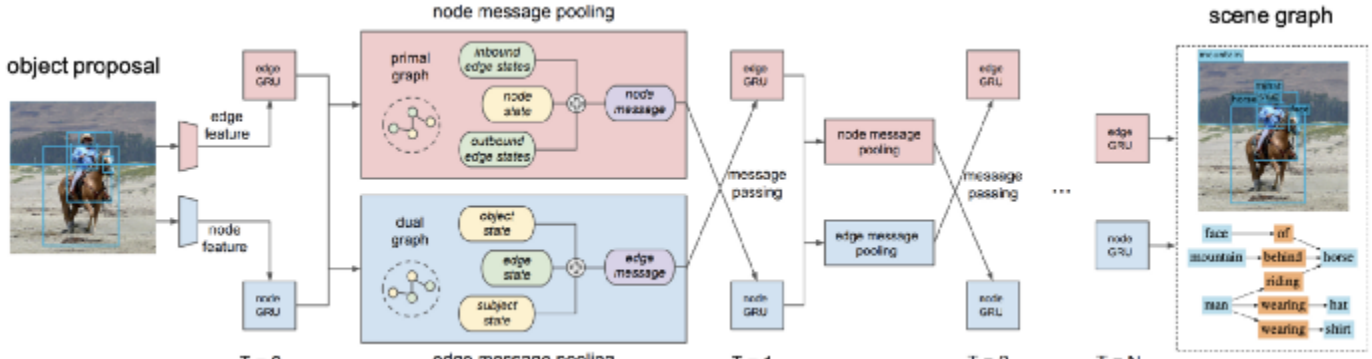






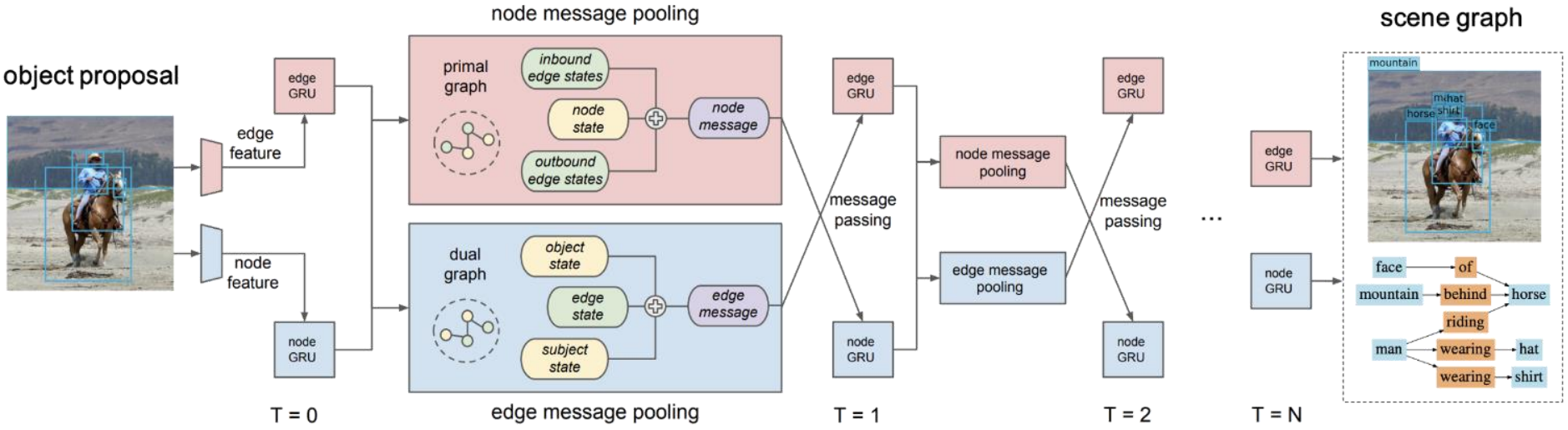


https://cs.stanford.edu/~danfei/scene-graph/



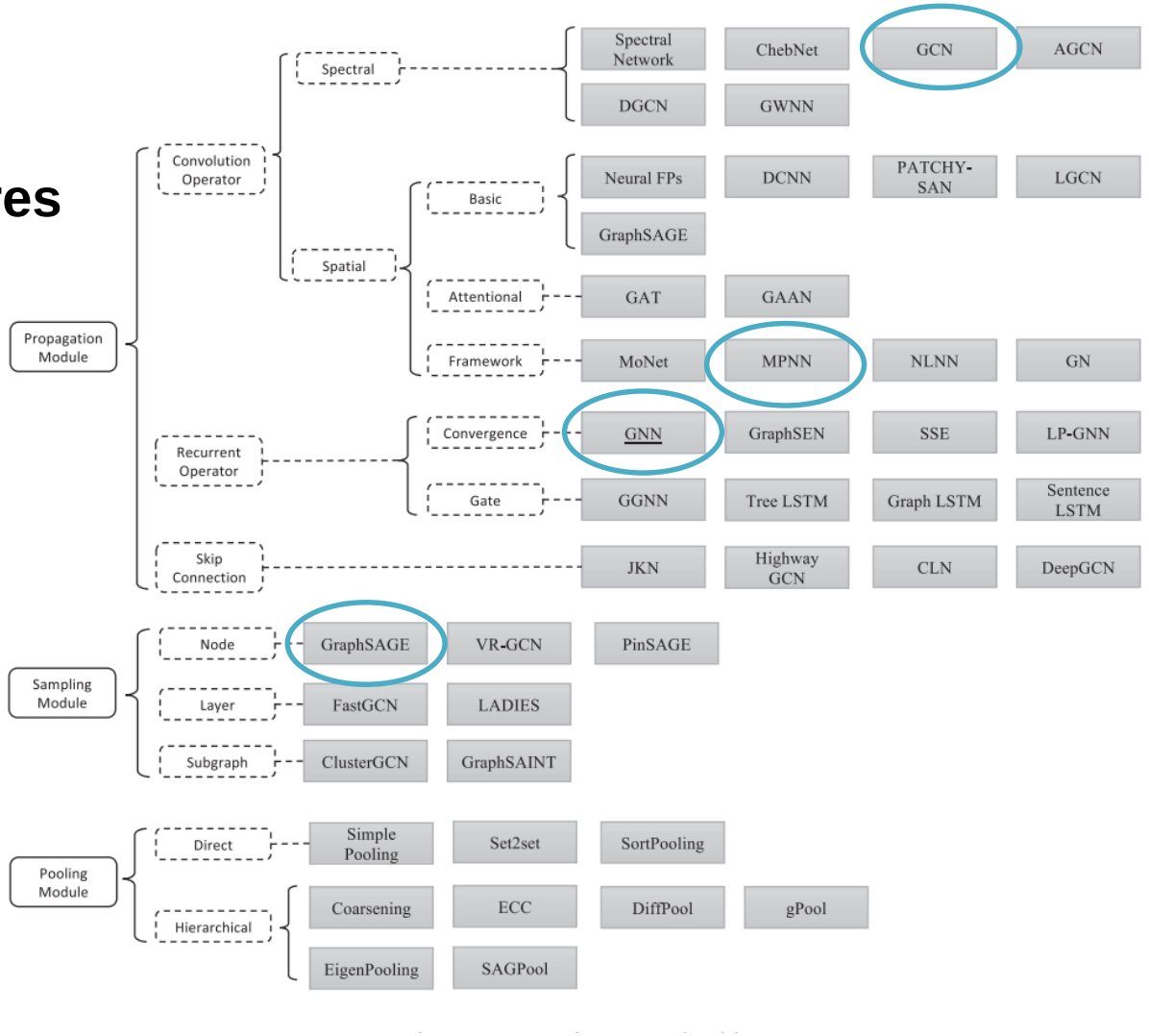
Applications

Scene Understanding



https://cs.stanford.edu/~danfei/scene-graph/

Additional Architectures



- (J. Zhou et al., 2020)

Knowledge Graphs: From Data to Wisdom
Prof. Dr. Markus Lienkamp

(Fabian Netzler, M. Sc.)

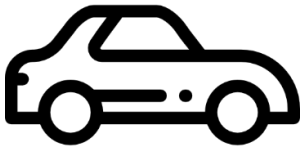
Agenda

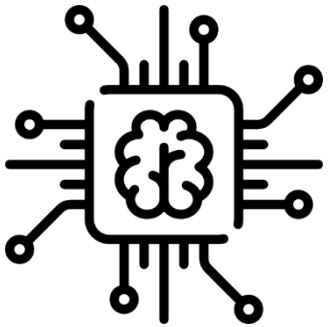
1. Chapter: Introduction into Graphs

2. Chapter: Path Planning
    - 2.1 Depth/Breadth Search
    - 2.2 Dijkstra

3. Chapter: Learning over Property Graphs
    - 3.1 Node Embedding
    - 3.2 Graph Neural Networks

4. Chapter: Summary





Summary

What we learned today:

Graphs can be used for path planning

Best/Deep/Breath search find paths in general

Dijkstra uses a heuristic to find a optimal path

Graphs can be extendend to be function as a general data structure

Graph embedding maps nodes to a vector space

The SkipGram algorithm uses the graph structure to map nodes

Graph Convolutional Netural Nets use node features

Message Passing Neural Nets can use edge features in addition