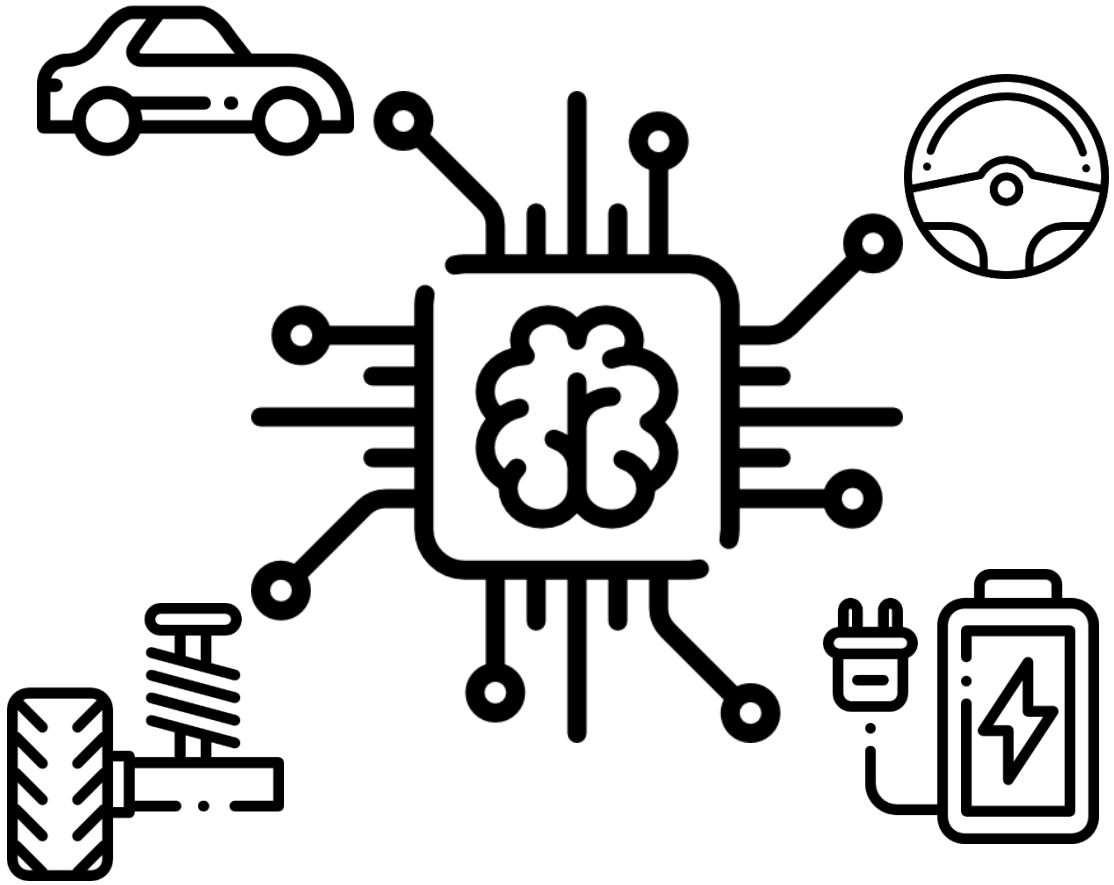
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Artificial Intelligence in Automotive Technology

Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann

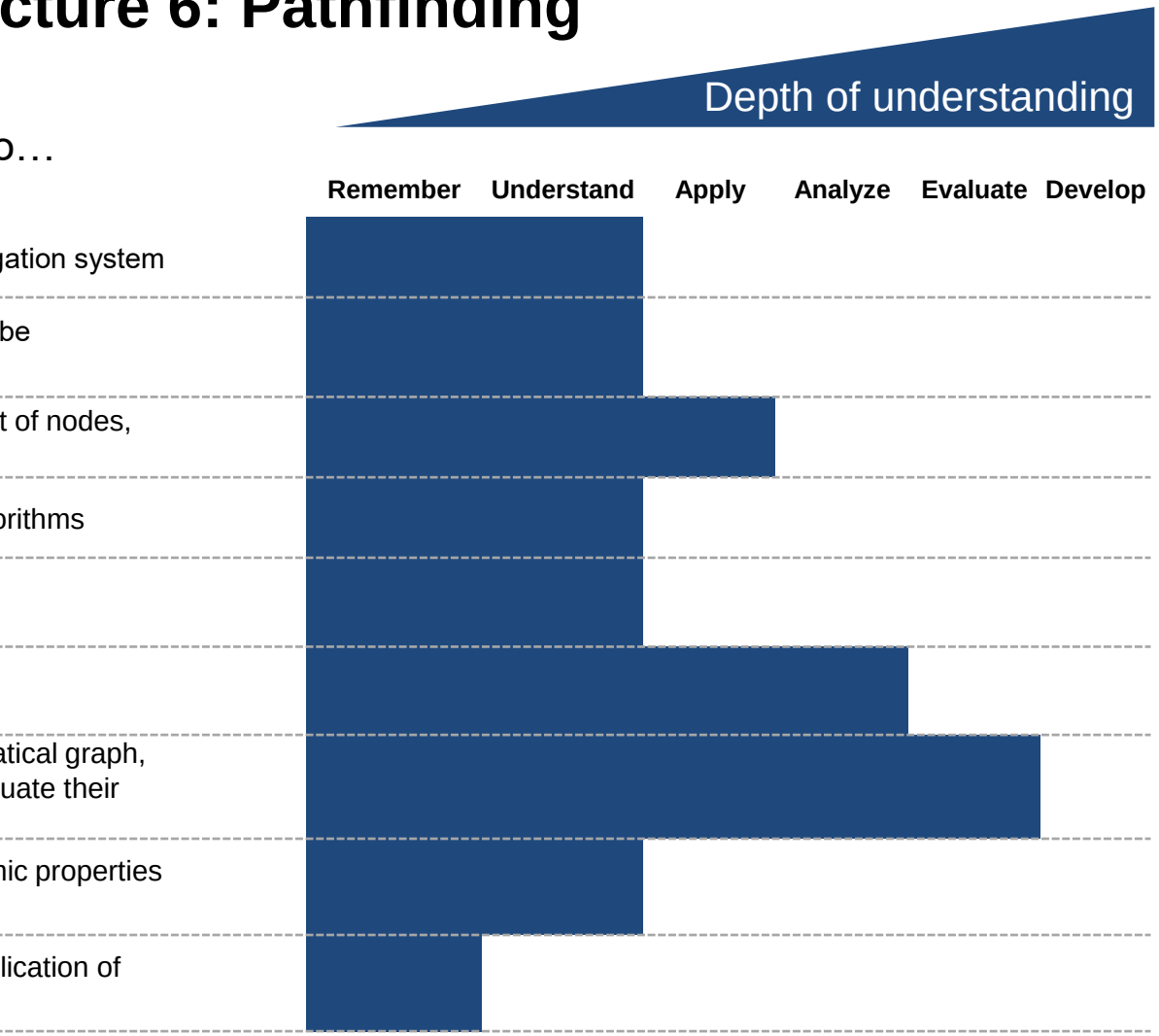


Lecture Overview

Lecture 16:15 – 17:45	Practice 17:45 – 18:30
1 Introduction: Artificial Intelligence 17.10.2019 – Johannes Betz	**Practice 1** 17.10.2019 – Johannes Betz
2 Perception 24.10.2019 – Johannes Betz	**Practice 2** 24.10.2019 – Johannes Betz
3 Supervised Learning: Regression 31.10.2019 – Alexander Wischnewski	**Practice 3** 31.10.2019 – Alexander Wischnewski
4 Supervised Learning: Classification 7.11.2019 – Jan Cedric Mertens	**Practice 4** 7.11.2019 – Jan Cedric Mertens
5 Unsupervised Learning: Clustering 14.11.2019 – Jan Cedric Mertens	**Practice 5** 14.11.2019 – Jan Cedric Mertens
6 Pathfinding: From British Museum to A* 21.11.2019 – Lennart Adenaw	**Practice 6** 21.11.2019 – Lennart Adenaw
7 Introduction: Artificial Neural Networks 28.11.2019 – Lennart Adenaw	**Practice 7** 28.11.2019 – Lennart Adenaw
8 Deep Neural Networks 5.12.2019 – Jean-Michael Georg	**Practice 8** 5.12.2019 – Jean-Michael Georg
9 Convolutional Neural Networks 12.12.2019 – Jean-Michael Georg	**Practice 9** 12.12.2019 – Jean-Michael Georg
10 Recurrent Neural Networks 19.12.2019 – Christian Dengler	**Practice 10** 19.12.2019 – Christian Dengler
11 Reinforcement Learning 09.01.2020 – Christian Dengler	**Practice 11** 09.01.2020 – Christian Dengler
12 AI-Development 16.01.2020 – Johannes Betz	**Practice 12** 16.01.2020 – Johannes Betz
13 Guest Lecture: VW Data:Lab 23.01.2020 –	

Objectives for Lecture 6: Pathfinding

After the lecture you are able to…

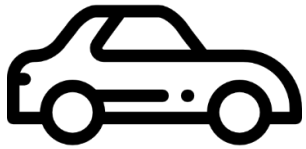


Pathfinding: From British Museum to A*

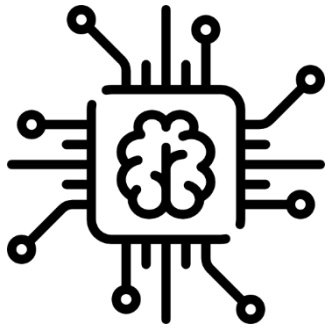
Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Lennart Adenaw, M. Sc.)

Agenda

1. Chapter: Introduction
    - 1.1 Navigation
    - 1.2 Digital Maps



2. Chapter: Algorithms
    - 2.1 British Museum Algorithm
    - 2.2 Breadth First, Depth First, Best First
    - 2.3 Dijkstra, A*

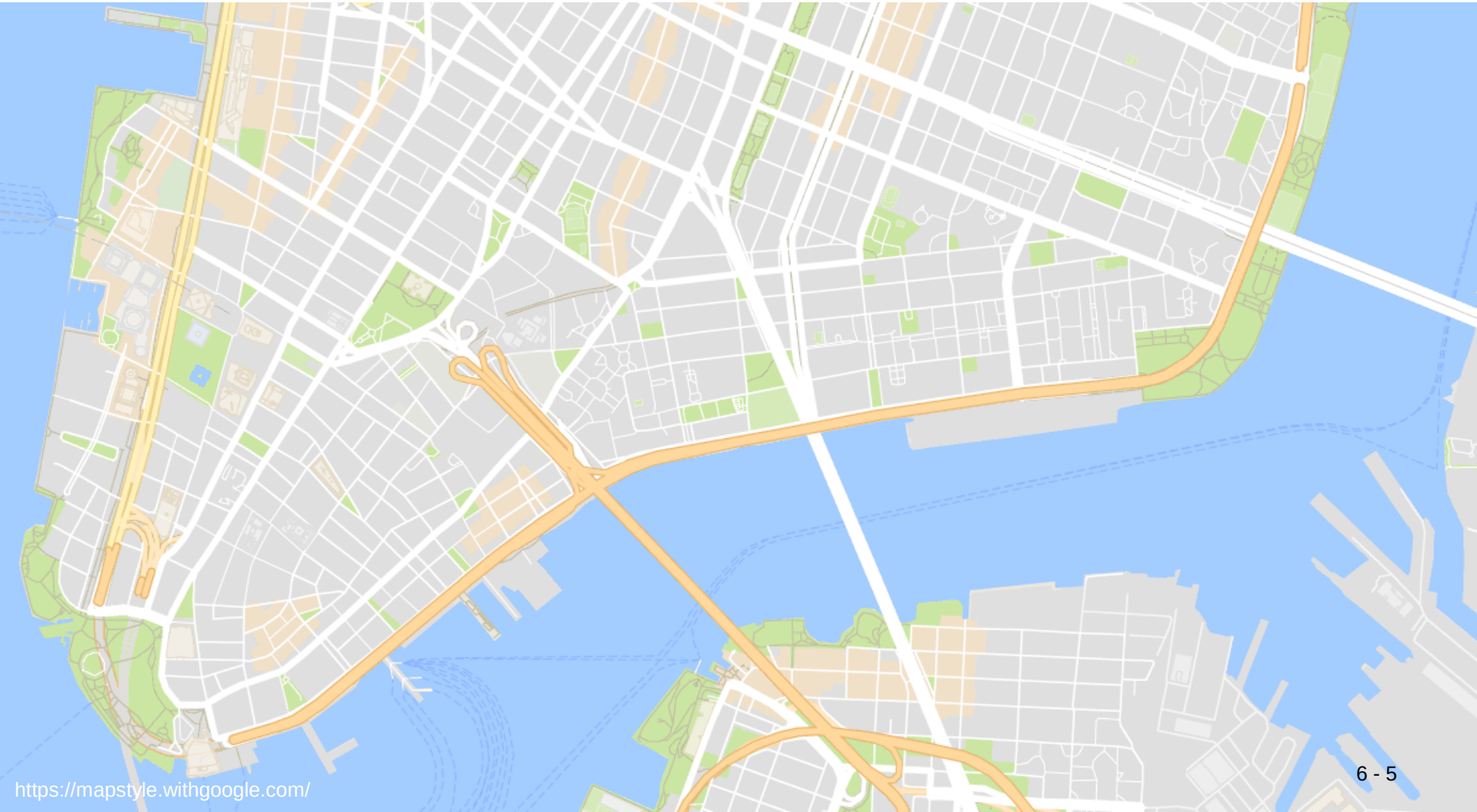


3. Chapter: Application

4. Chapter: Summary

Introduction – Navigation

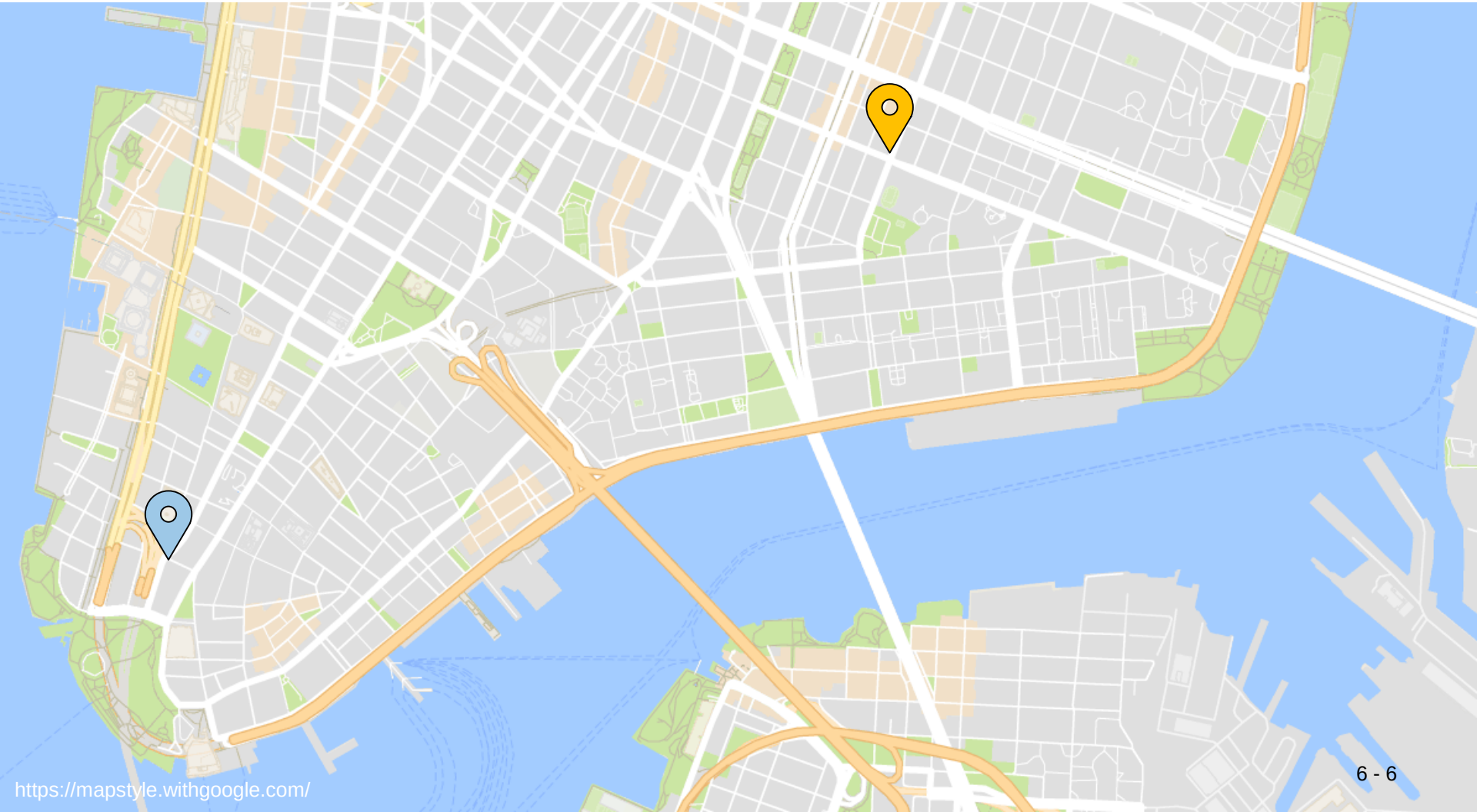
The human approach



https://mapstyle.withgoogle.com/

Introduction – Navigation

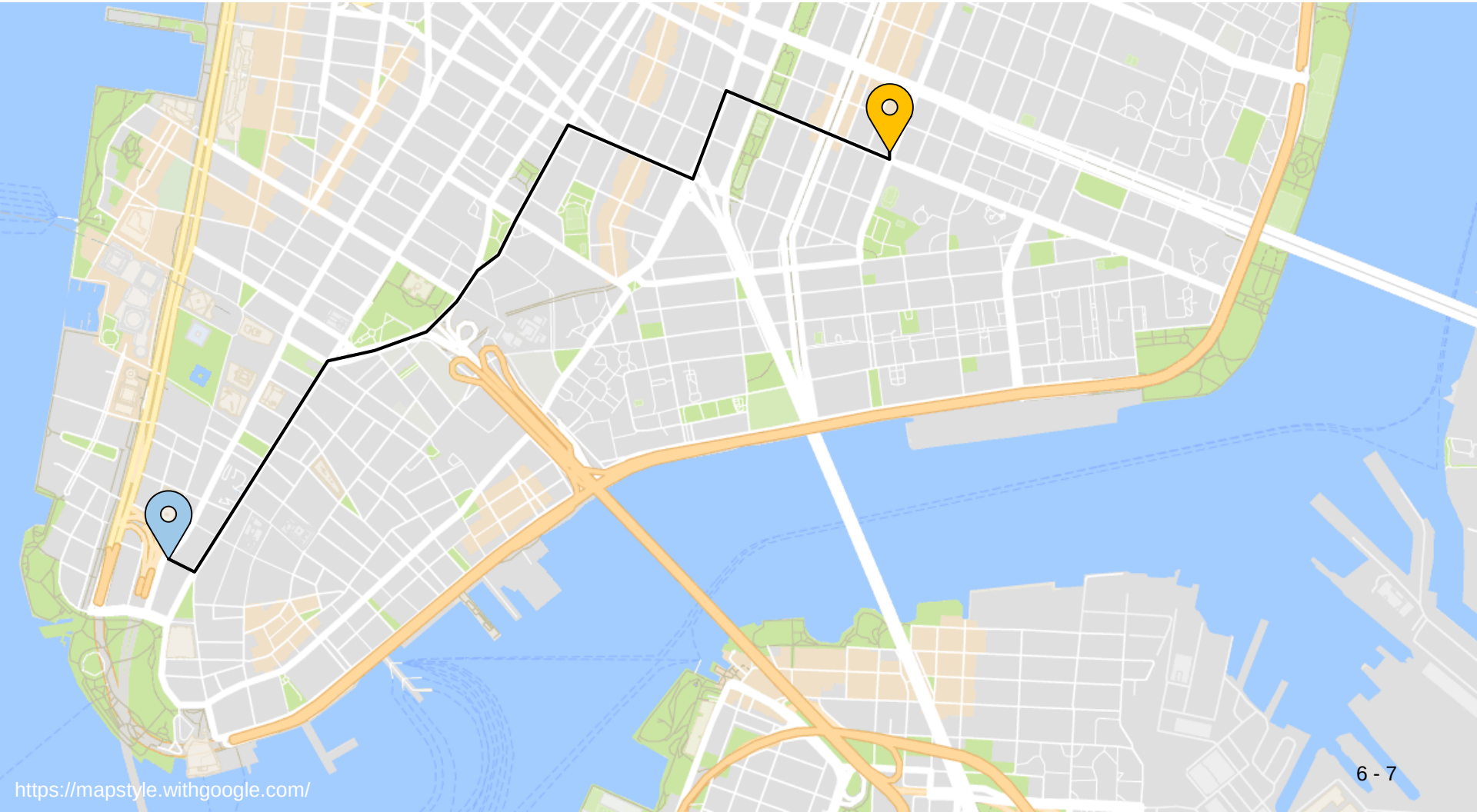
The human approach



https://mapstyle.withgoogle.com/

Introduction – Navigation

The human approach



https://mapstyle.withgoogle.com/

Additional Slides

The human approach to finding routes on a street map is a very visual one. Feasible paths are developed by visually tracing different street lines in the general direction of the target. Obstacles, dead ends and restrictions are intuitively taken into account, although – in typical routing situations – no intensive thinking or reflection is carried out. The human act of finding a path between a starting point and a destination can be considered as „intelligent“.

As a matter of fact, we do not understand the human approach of finding routes on maps in detail. Hence, we cannot model it directly. Instead, search algorithms have been evolved that solve the same problems by supposedly different means. Some popular ones of these shall be introduced in this lecture.

The question that remains is whether or not the presented algorithms can themselves be considered as „intelligent“ solely because they solve a problem that humans solve by means of human intelligence.

Introduction – Navigation

Definition

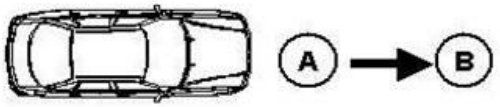
„The process or activity of accurately ascertaining one's position and planning and following a route.”

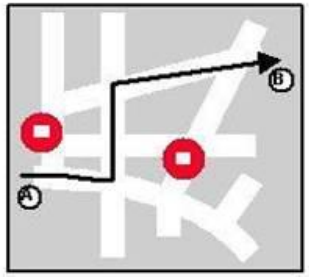
https://en.oxforddictionaries.com/definition/navigation

Introduction – Navigation

Definition

1. Navigation



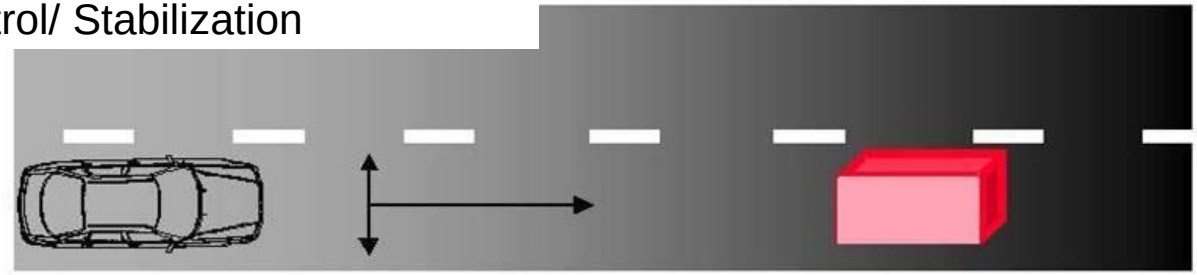


Hours to Minutes

2. Path Following

Minutes to Seconds

3. Control/ Stabilization



Seconds to Miliseconds

Additional Slides

The overall task of driving a car can be separated into three sub-tasks or „levels of driving“.

1. **Navigation**

   The first level of driving comprises the Navigation-task. Navigation – as defined a few slides earlier – is the „process or activity of accurately ascertaining one's position and planning and following a route”. However, when it comes to levels of driving, Navigation solely refers to the planning process. During Navigation, the driver determines his position and destination and plans a route from the first to the latter. Outputs of this phase are paths/routes that can change during the driving task due to additional information (e.g. traffic). Therefore, the timeframe of the Navigation-level can be said to be hours to minutes.

2. **Path Following**

   During driving, the driver seeks to follow the path determined on the Navigation-level. This task is referred to as Path Following. To follow the designated path, actions like turns and lane changes may have to be conducted within minutes to seconds.

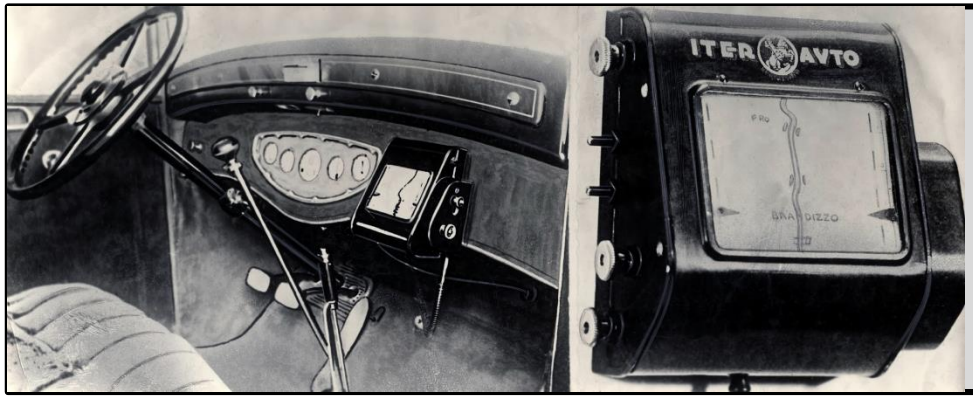
3. **Control/Stabilization**

   The most time-critical driving level is the Control/Stabilization level. On this level, high frequent speed choices, braking and obstacle evasion are made. Control/Stabilization is characterized by a very dynamic time horizon of seconds to milliseconds.

This lecture focuses on the Navigation-level of driving. Hence, we will be looking at technical means to replace human reasoning in route planning and aid the driver during Path Following.

Introduction – Navigation

Historical Approaches



Paper Map
Direct Transmission
„Play“ Specific Paths

1930 Iter Avto

Paper Map
Gyroscope
Scrolls Map

1981 electro Gyrocator

https://ndrive.com/brief-history-gps-car-navigation/

Introduction – Navigation

Historical Approaches

Digital Map
Dead Reckoning
Cassettes

1985 Etak Navigator

Digital Color Map
Dead Reckoning
CD-ROM

1987 Toyota CD-ROM Navigation System

https://ndrive.com/brief-history-gps-car-navigation/

Introduction – Navigation

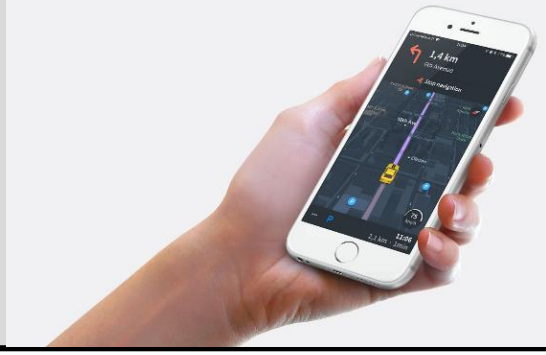
Historical Approaches

Digital Color Map
GPS

1990 Mazda Eunos Cosmo

GPS
Dead Reckoning
Map Matching
Speech Recognition
Dynamic Routing
Points of Interest (POI)

Modern Navigation Systems



https://ndrive.com/brief-history-gps-car-navigation/

Additional Slides

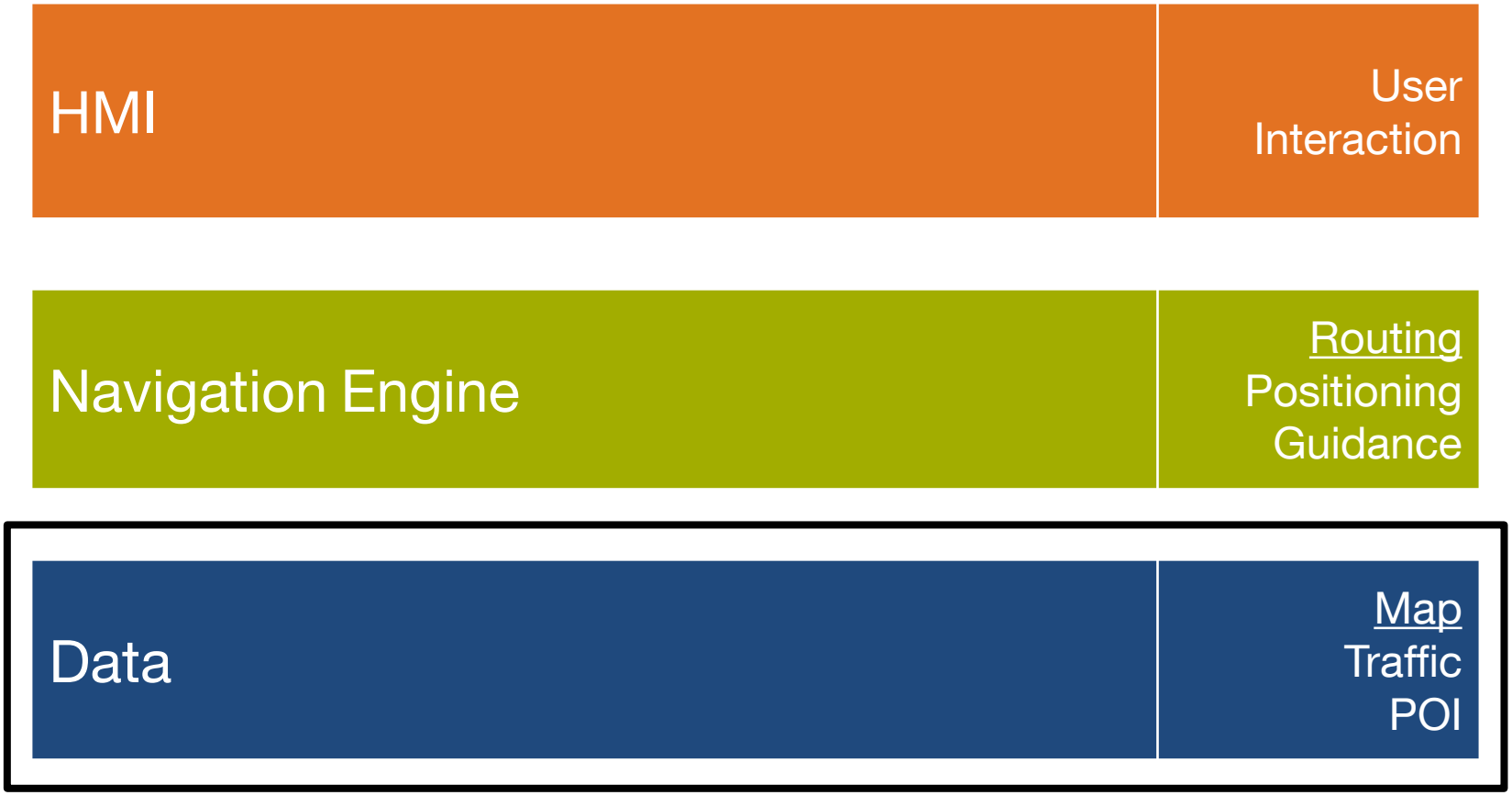
Navigation systems have been around since the early 1930s. The 1930 Iter Avto unwound a paper roll showing the path, by means of a mechanical transmission. The paper had to be labeled prior to driving with the specific path a driver wanted to use. Hence, Iter Avto did not address all tasks of navigation. The planning was done by a human and the system only guided in positioning and path following. A brainchild of the Iter Avto were systems like the electro Gyrocator from 1981, which automatically scrolled a physical map according to the car‘s movements. Again, these systems did not solve the planning task. Nevertheless, they helped with positioning and aided path following. From the mid-80s, fully functional navigation systems emerged. These systems use digital maps, modern planning algorithms and electronic positioning systems (either GPS or dead reckoning).

However big the differences: a common denominator of historical and present navigation systems is the usage of a map for navigation. A map, according to the oxford dictionary, is „a diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc.“.

Hence, it constitutes a model of the physical appearance of reality. In the context of car navigation, maps are mainly used to model the street network. Navigation can then be conducted using this model. Thanks to GPS-sensors, navigation in modern navigation systems is technically reduced to route planning on the basis of destination information and position sensing. For the scope of this lecture, we therefore assume the subtask of „ascertaining one‘s position“ to be a solved issue, handled by an integrated sensor, directly delivering a GPS-position to the planning instance.

Introduction – Navigation

What is needed for car navigation?



Additional Slides

There are three basic parts of a modern navigation system:

The **HMI** (Human-Machine-Interface) is used for user interaction. Information is passed through the HMI from the user to the navigation system. For example, the user uses the HMI to input the destination he wants to drive to.

The **Navigation Engine** takes care of positioning and pathfinding. In order to solve these tasks, it utilizes GPS sensors, information from the **Data Base** – including the digital map – and pathfinding algorithms, which will be introduced throughout this lecture.

The **Data Base** provides the digital map data, as well as additional information. This information is manifold and differs from vendor to vendor. Most often, the data base also contains information on place-names in different languages, POI (Points of Interest), street conditions, speed limits, traffic, weather and much more.

The HMI will not be investigated during this lecture. Instead, we will address digital maps as a basis for path planning and path planning algorithms, hence, focusing on the technical core of the car navigation process.

The upcoming slides will introduce you to the concepts of digital maps.

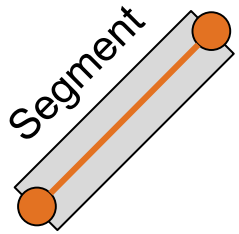
Introduction – Digital Maps

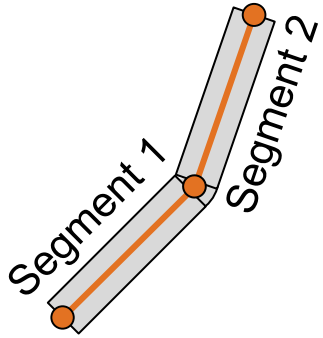
What needs to be modelled?

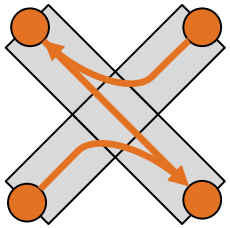
Road Geometry
- Position
- Length
- Start - End

Road Connectivity
- Intersections
- Connections
- Fly-Over

Road Attributes
- Turn Restrictions
- Speed Limits
- Type







Additional Slides

The basis of any pathfinding task is the map. As introduced before, navigation systems exclusively employ digital maps nowadays. Therefore, it is of paramount importance to be able to represent street networks by means of a digital data model prior to using routing algorithms to find suitable connections between origin and destination.

In order to successfully reproduce physical street networks for the use in a digital navigation engine, at least three basic categories of road properties have to be modelled:

1. **Road Geometry**
   The most important information when it comes to modeling street segments is the geometry of a segment. Each segment is assigned a start and endpoint, determining its length and position.
2. **Road Connectivity**
   Besides Road Geometry, Road Connectivity information is needed to distinguish flyovers and bridges from connected street segments. This is done by sharing points.
3. **Road Attributes**
   Road Attributes like turn restrictions, speed limits and street type are needed to determine feasible and optimal (duration, length) paths from origin to destination and take care of restrictions during pathfinding (avoid highways, tolls, etc.).

Equivalent to paper maps, Road Geometry and Road Connectivity can be modelled using two basic elements: **points** and **lines**. This approach resembles the way you would sketch a map by hand. The missing information (Road Attributes) can then be added using **tags**, which are linked to these elements and describe them further.

The basic idea of representing digital maps using points, lines and tags has been picked up by many map suppliers, of which Open Street Maps (OSM) shall be presented in this lecture. The OSM data model is introduced in the following slides.

Introduction – Digital Maps
OSM

Open Steet Map (OSM)

- Worldwide Map Data
- Free
- Started 2004 in London
- 1 000 000 Contributing Users
- Defines **Data Model** for Mapping

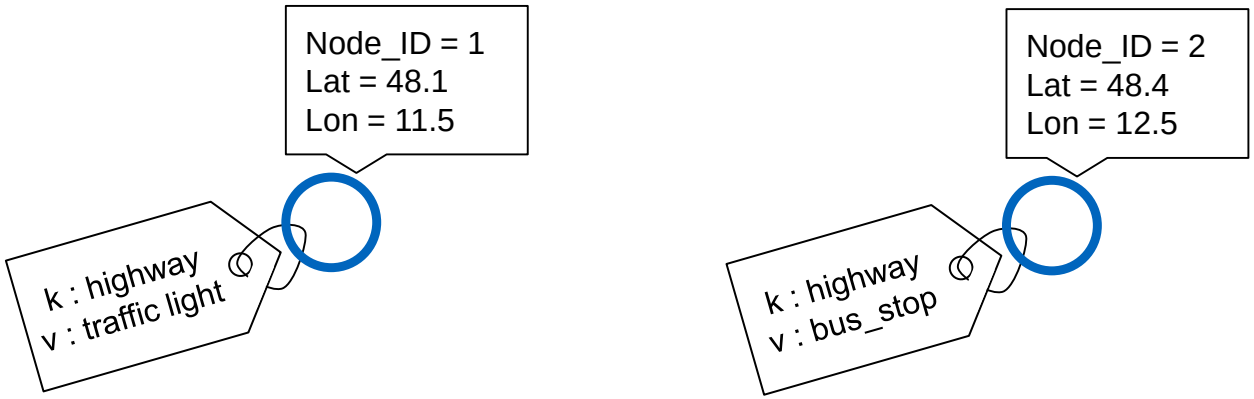
https://upload.wikimedia.org/wikipedia/commons/thumb/b/b0/Openstreetmap_logo.svg/1200px-Openstreetmap_logo.svg.png

Introduction – Digital Maps

OSM

Nodes

Defined by latitude, longitude and node id. Represents arbitrary locations.
Attributes specified by tags.



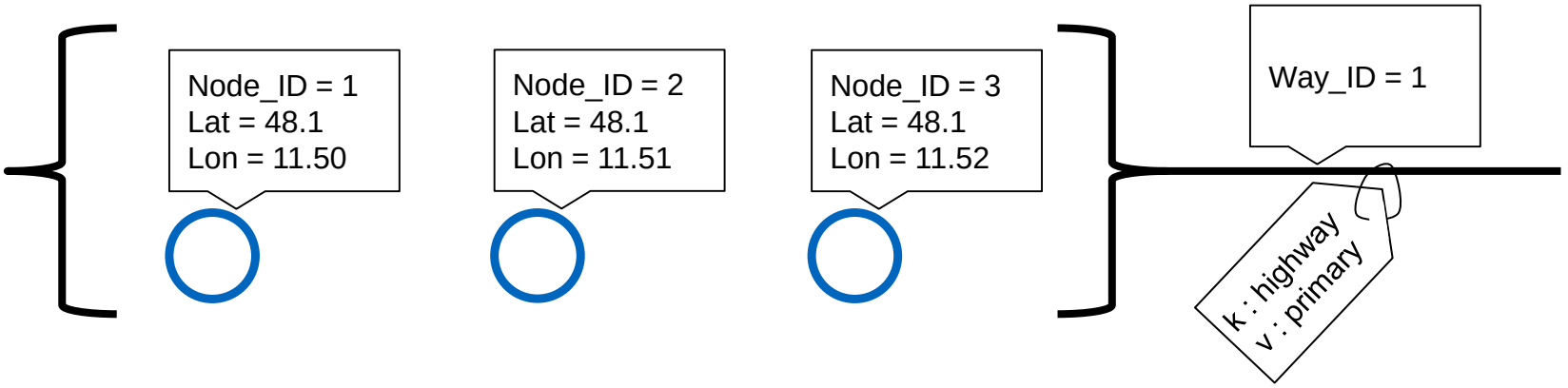
A **tag** consists of a **key** and a **value**. Tags describe features of map elements.
Keys and values are free text but conventions exist.

Introduction – Digital Maps

OSM

Ways

Defined by an ordered set of nodes. Groups nodes into lines.
Attributes specified by tags.



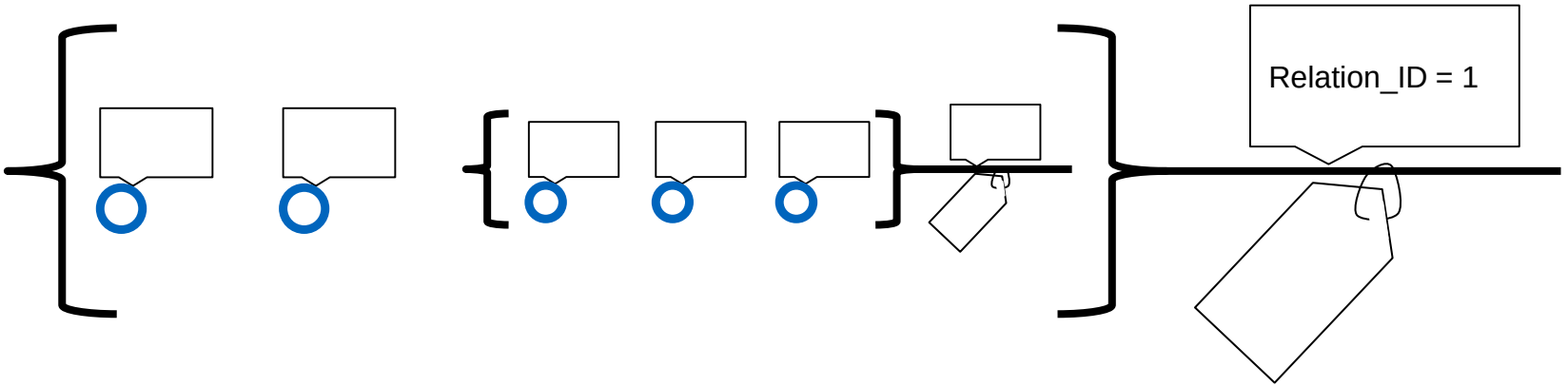
A **tag** consists of a **key** and a **value**. Tags describe features of map elements.
Keys and values are free text but conventions exist.

Introduction – Digital Maps

OSM

Relations

Defined by an ordered list of nodes, ways and/or relations. Defines logical or geographic relationships between other elements.



A **tag** consists of a **key** and a **value**. Tags describe features of map elements.
Keys and values are free text but conventions exist.

Additional Slides

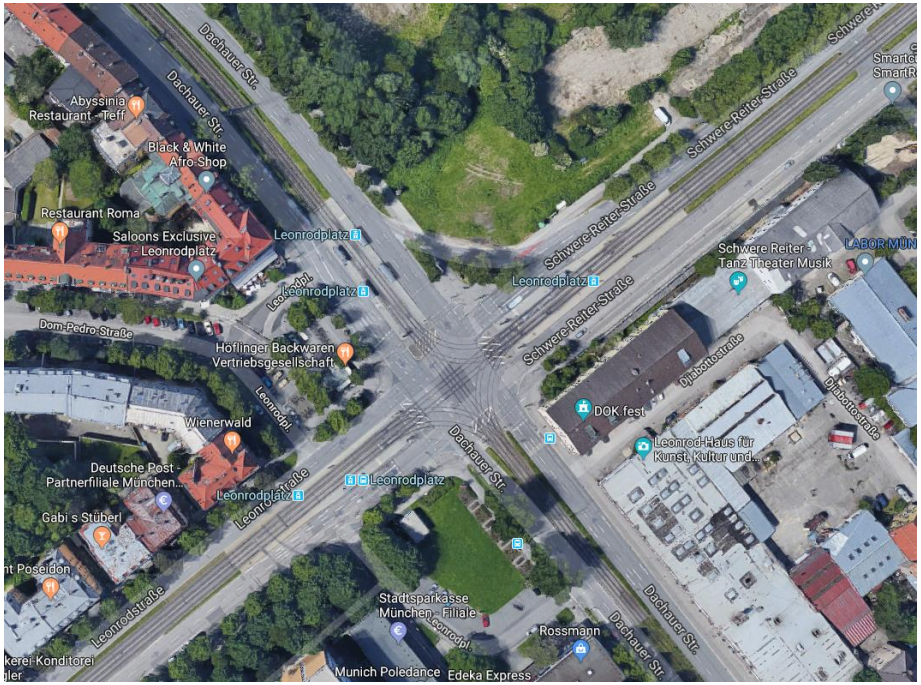
OSM defines a data model to effectively implement the basic ideas of street modelling in practice. Points are called **Nodes** in OSM and lines are known as **Ways** in the OSM jargon. In addition to these geometric concepts, **Relations** model real-life relationships that are not physically observable or built upon existing Nodes and Ways in a specific way. Examples for relations are bus routes and hiking trails (that are composed of different roads already in the database).

All of the elements above may be assigned a set of **Tags** in order to describe their attributes (names, operators, number of lanes, type of the street, etc.).

Introduction – Digital Maps

Data Sources

Satellite Imagery



https://www.maps.google.de

GPS Tracks

https://buy.garmin.com/de-DE/DE/p/550460

Additional Slides

A data model for digital maps alone is of no practical use. The main value of a digital map is the information stored inside the data model. This data has to be collected from various sources, in order to bring live to a digital map data base. In the case of OSM, a huge number of people contribute to the project voluntarily by mapping streets, buildings, POI, landmarks, etc.. They do so by defining and tagging the nodes and ways that best describe their physical counterparts. For this work, they utilize GPS tracking devices to record positions or interpret satellite imagery.

In addition to this, land surveying offices provide the data they collected and private companies may also decide to contribute by feeding their data bases into the OSM data base.

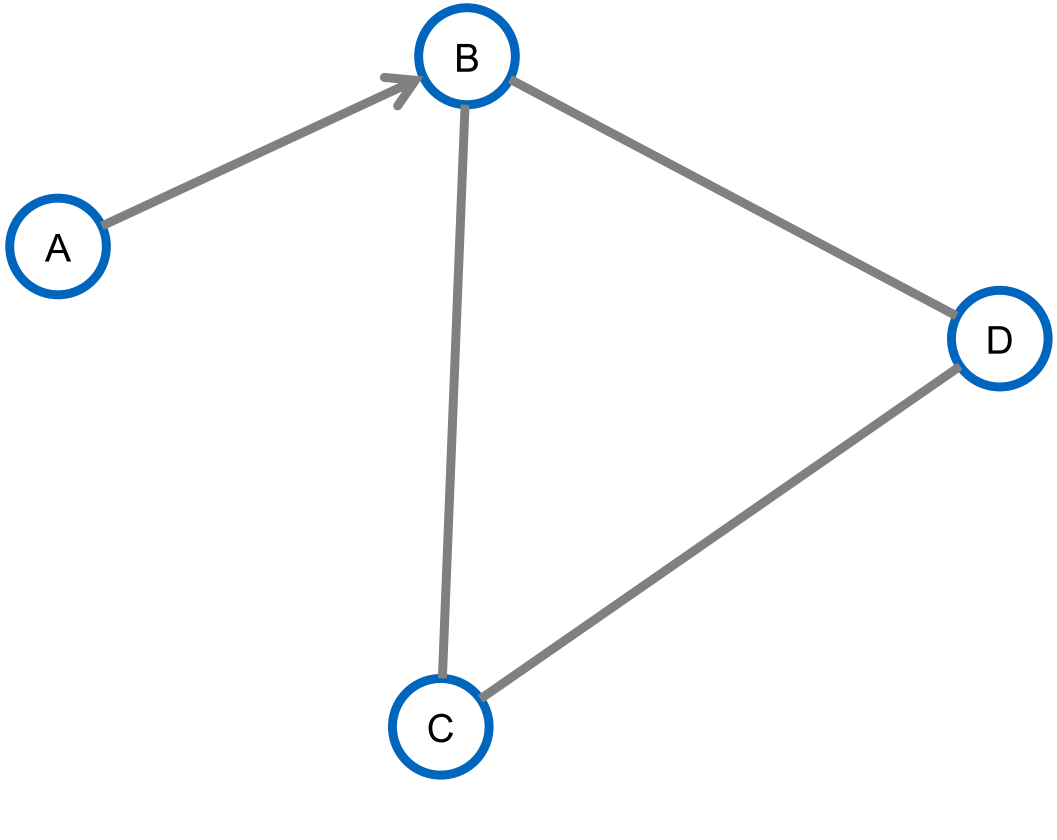
Introduction – Digital Maps

Graphs – Basic Elements

Node

Directed Edge

Undirected Edge



Additional Slides

The data models for digital maps are structurally related to mathematical graphs. Mathematical graphs in turn can easily be formalized and used for automatic processing by pathfinding algorithms. Due to this reason, they constitute a link between mapping and pathfinding. Navigation engines are built upon mathematical graphs that are calculated from the digital map. The graph ultimately used for routing is restricted to the information indispensable for the pathfinding tasks.

In graphs, ways (streets) are cut into segments called edges. Therefore, the concept of a street is suspended, because streets – in the sense of collections of segments – are not directly relevant to the pathfinding tasks. Consequently, streets segments without crossroads or speed limit changes are oftentimes represented by a single edge in the routing graph. The reason for this is that the driver can only use or not use the street segments corresponding to the edge, but once he enters any of these segments he cannot do anything besides following the street until the next crossroads enables him to change direction. From a routing point of view, it is therefore redundant to sustain all street segments in the digital map when generating a routable graph. Hence, nodes in a routing graph correspond to crossroad positions or changes in road attributes that are relevant to the pathfinding process.

In graphs, there are two types of edges: directed and undirected ones. Directed edges can only be traversed in one direction, hence modeling one-way streets or individual lanes, whereas undirected edges work both ways.

Introduction – Digital Maps

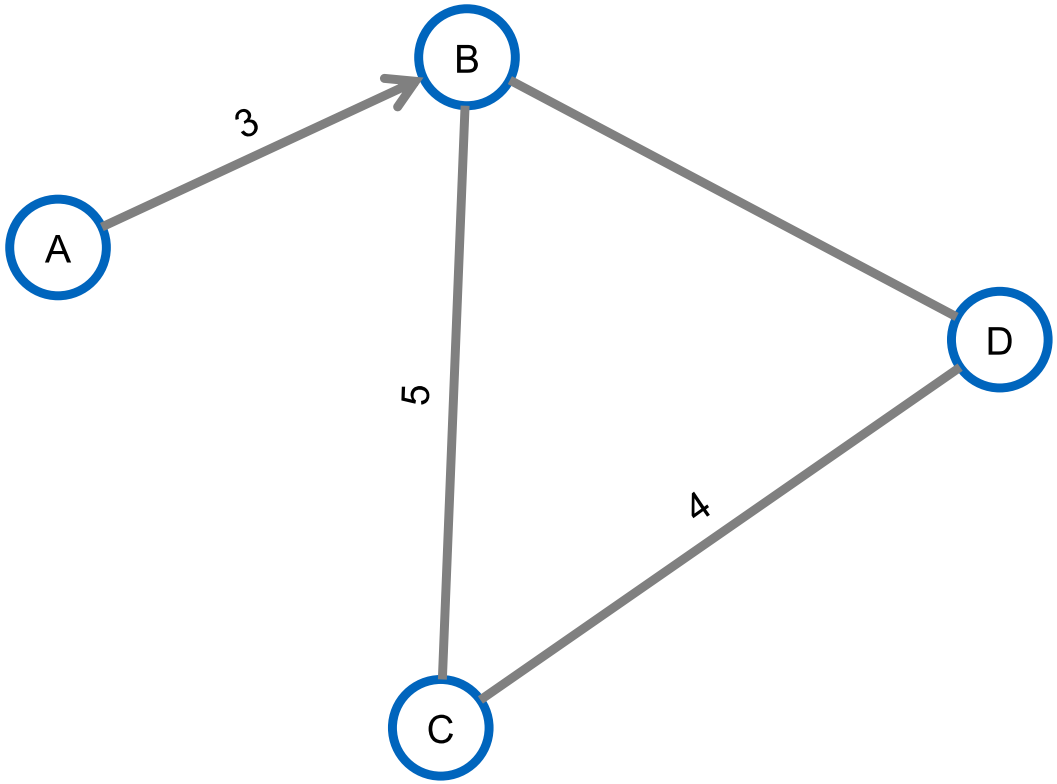
Graphs – Basic Elements

Weighted Edge

3

Unweighted Edge

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



Additional Slides

Oftentimes, a graph is not used exclusively to find an arbitrary path through the street network, but to find the most cost efficient way to do it. Any pathfinding algorithm optimizing costs needs cost information stored in the graph. Storing cost information in a mathematical graph is possible using so-called edge weights. An edge weight is associated with an edge and denotes the cost of traversing that edge a single time.

In the context of navigation systems, costs can for example be units of time, street length or air pollution (CO_2 emission).

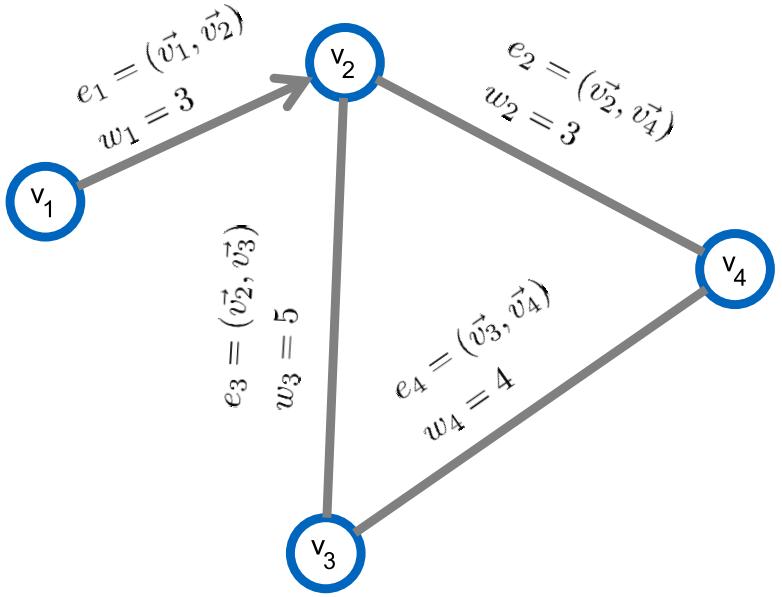
Introduction – Digital Maps

Graphs – Formal Description

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

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

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



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

Additional Slides

As mentioned before, graphs can be formalized conveniently. In consequence, they can be processed automatically, are unambiguous and efficient to store. The common notations used in graph theory are used in this lecture.

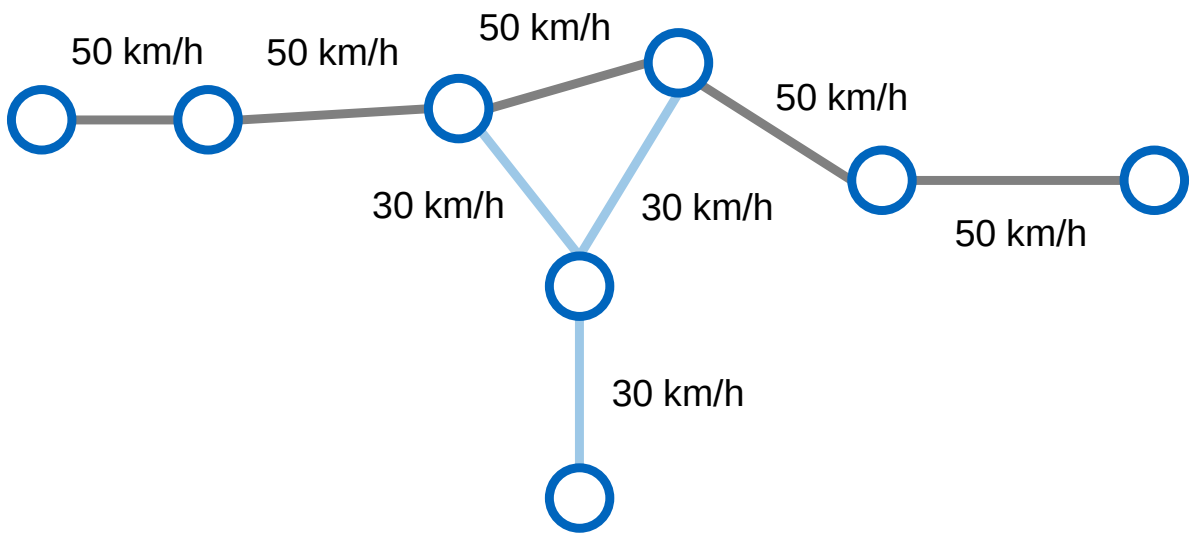
Building upon the knowledge from the previous slides, we will build an exemplary graph of a crossroads stored in the OSM data.

Introduction – Digital Maps

From OSM to Routable Graphs

Initial Graph

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

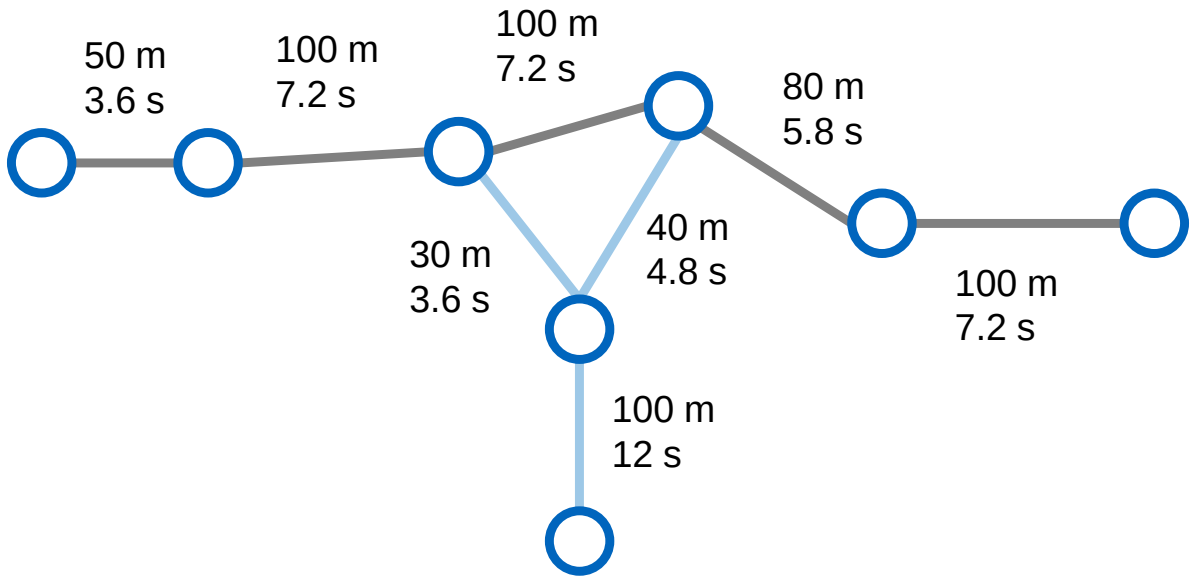


Introduction – Digital Maps

From OSM to Routable Graphs

Augmented Graph

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

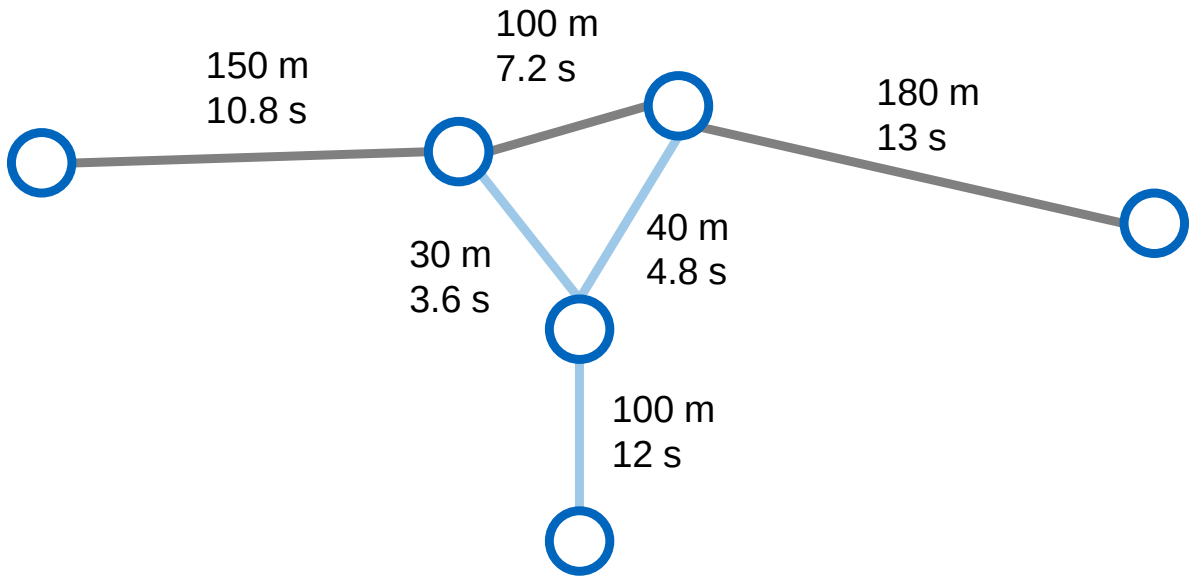


Introduction – Digital Maps

From OSM to Routable Graphs

Compressed Graph

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

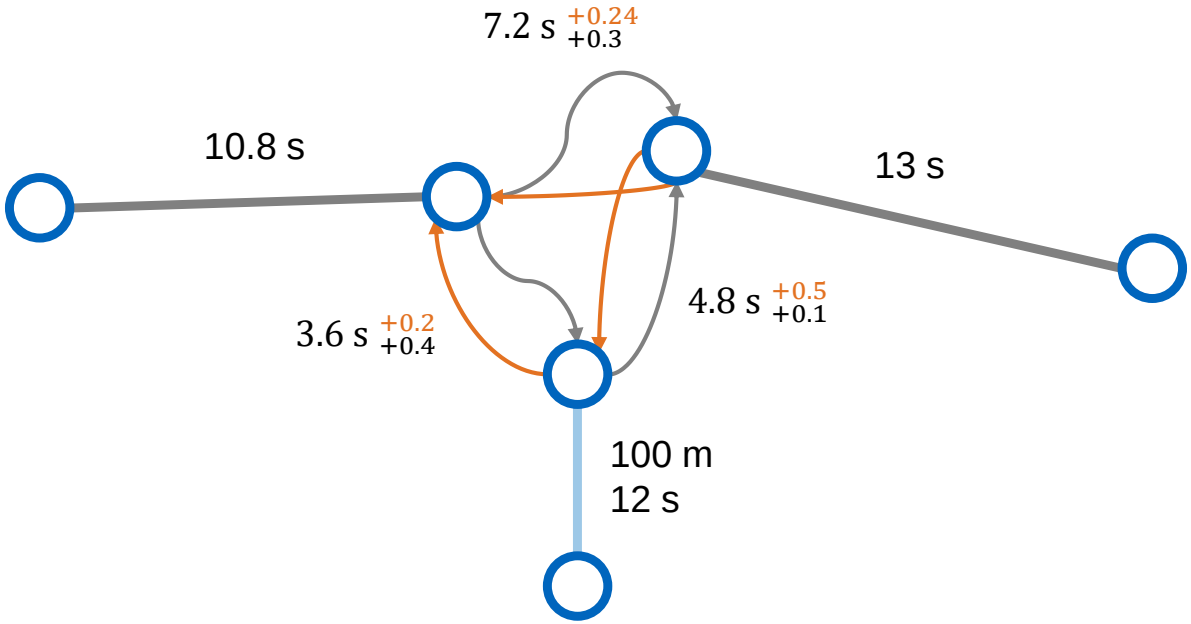


Introduction – Digital Maps

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



Additional Slides

During the graph generation, a digital map data model (like OSM) is translated into a graph for pathfinding. In this process, redundant nodes are removed (compression) and tags from the digital maps are interpreted to introduce necessary costs, turn restrictions and turn penalties to the graph. The resulting path is ready to be used for pathfinding.

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

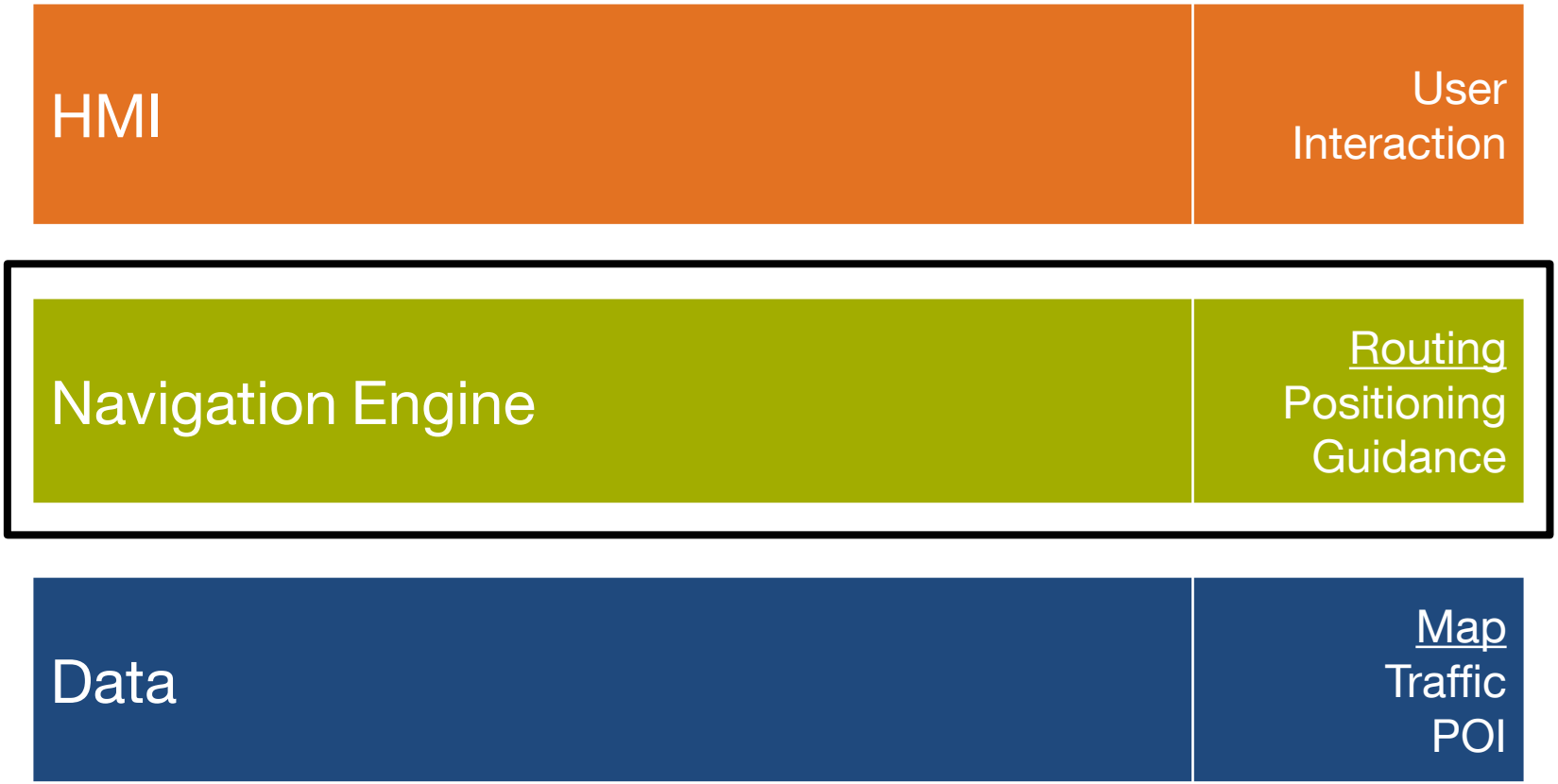
Additional Slides

Throughout this lecture, a set of assumptions is established for the sake of simplicity. These assumptions may not be valid in real-life applications. Nevertheless, there are adaptions to all pathfinding algorithms, which are introduced during this lecture, that guarantee success even if these assumptions do not hold. Any popular assumptions or enhancements of the algorithms presented in this lecture are easy to understand when knowing about the basic algorithms. This is why this lecture focuses on the core ideas rather than on the real-life peculiarities of the graphs or algorithms.

1. Non-negative edge weights
   Non-negative edge weights ensure that no situation is possible in which a pathfinding algorithm is rewarded for going in eternal circles, cumulating negative costs without finding the destination. Non-negative edge weights may be used in real-life to reward the (one-time) usage of specific street segments.
2. Edges are undirected
   In the remainder of this lecture, edges are always undirected, solely for the sake of simplicity. All pathfinding algorithms in this lecture work on graphs with directed edges as well. In real-life, directed edges can be used to model one-way streets and turn restrictions.
3. Nodes are given in a two-dimensional space
   Two dimensions are more convenient than working with multiple dimensions. In addition, most world coordinate systems employ two dimensions and the most intuitive interpretation of a map is a flat space. In reality, certain applications may involve the use of a third dimension for elevation.
4. Edges weights are defined using the Euclidean distance
   Edge weights in real-life applications may be used to reflect different measures like distances, durations, emissions etc. or combinations of these. The routing algorithms do not distinguish between the meaning of a measure, which is why the concepts are interchangeable. Hence, this lecture focuses on a distance cost approach measured by the euclidean distance, which is possible because of assumption 3.

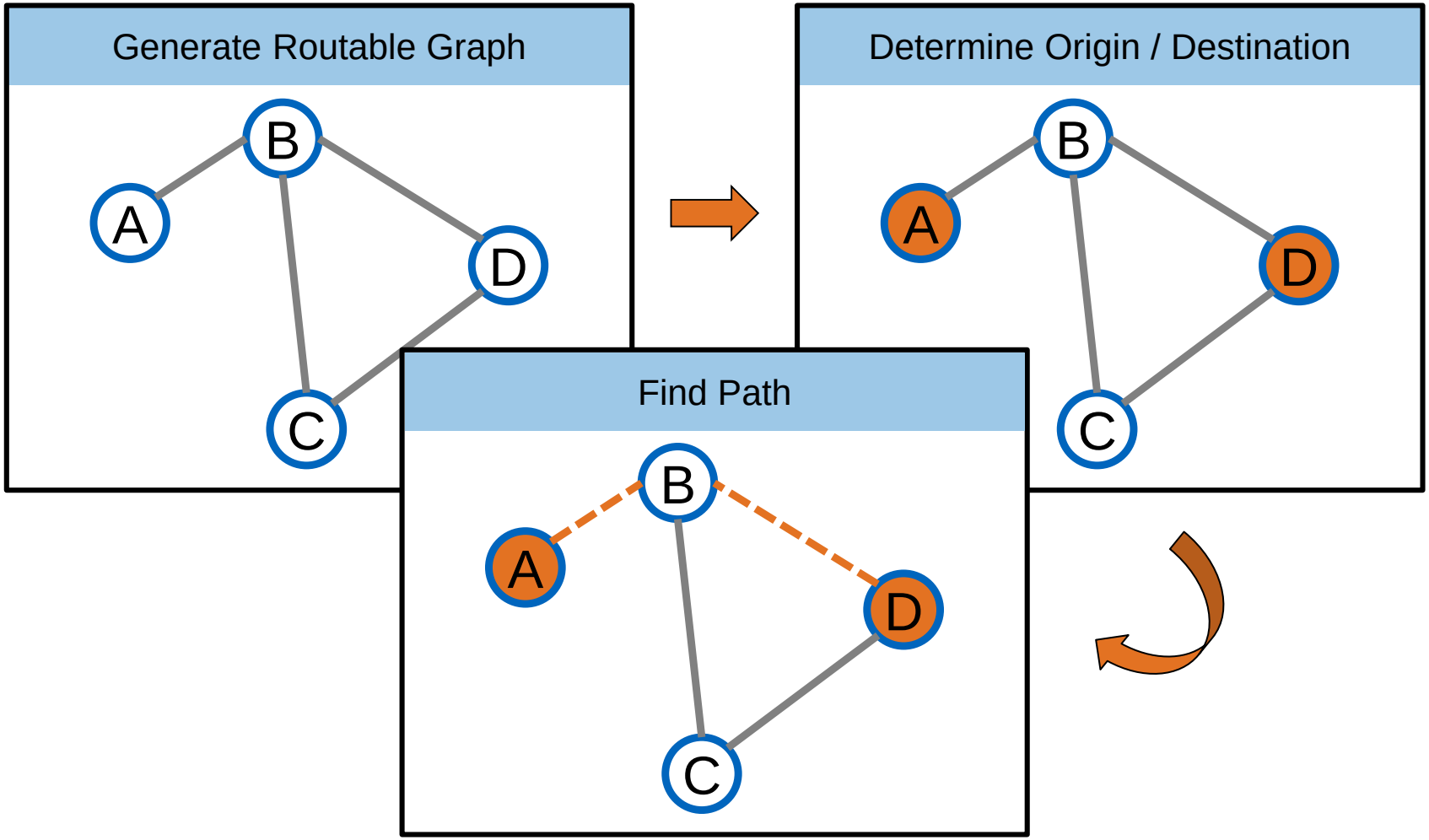
Introduction – Navigation

What is needed for car navigation?



Algorithms

Overview Routing

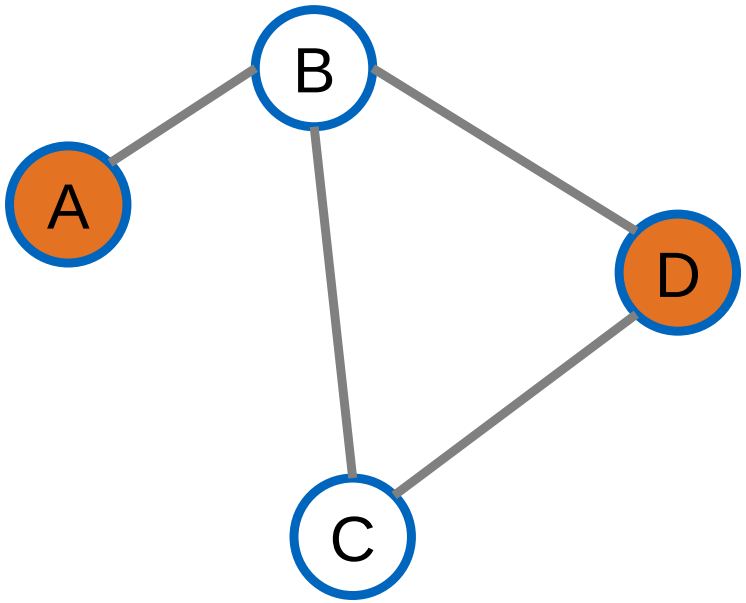


Additional Slides

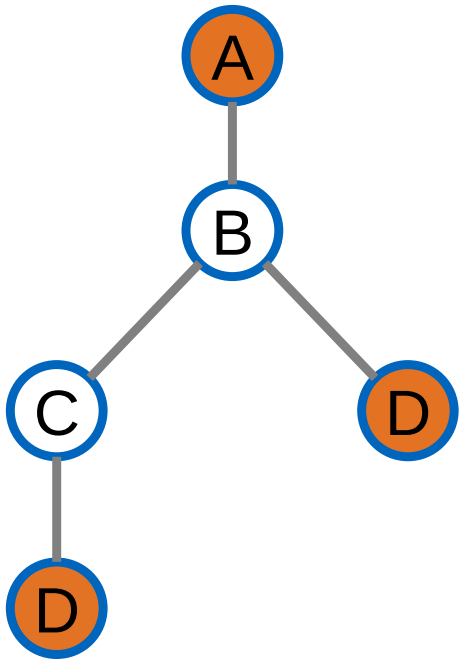
The overall task of routing a graph – e.g. finding a path through the graph – comprises three elementary steps. First of all, a graph has to be generated. Afterwards, origin and destination are determined in the graph and lastly, a path between these two points is determined by the use of a routing algorithm. Up to this point, we have covered graph generation and decided to determine the origin from the current position as supplied by a GPS sensor. The destination can then be entered via the navigation system’s HMI. The actual process of finding a suitable path is then left to pathfinding algorithms which are introduced in the upcoming slides.

Algorithms
Search Trees

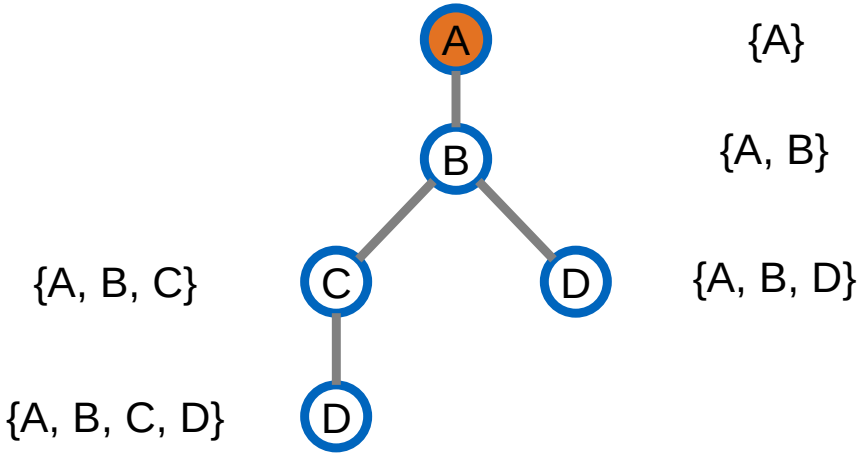
Graph



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.

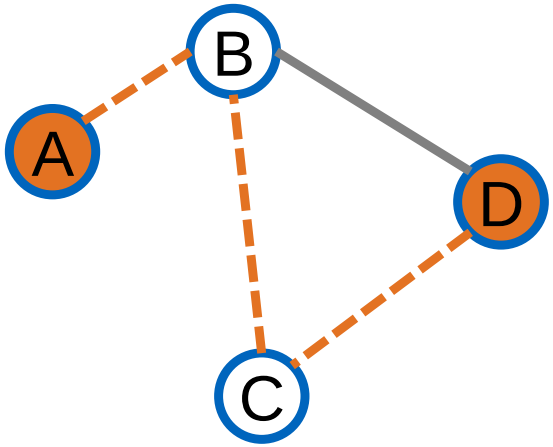
(Almost) all algorithms in this lecture

Algorithms

Pathfinding: Problem Formulations

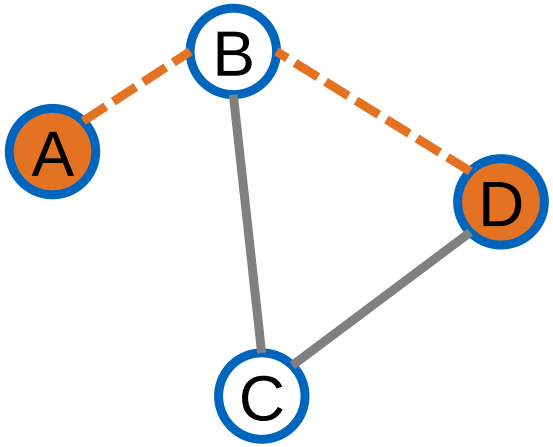
A : Find a Path

- Depth First
- Breadth First
- Best First



B : Find an optimal Path

- Dijkstra
- A*



Additional Slides

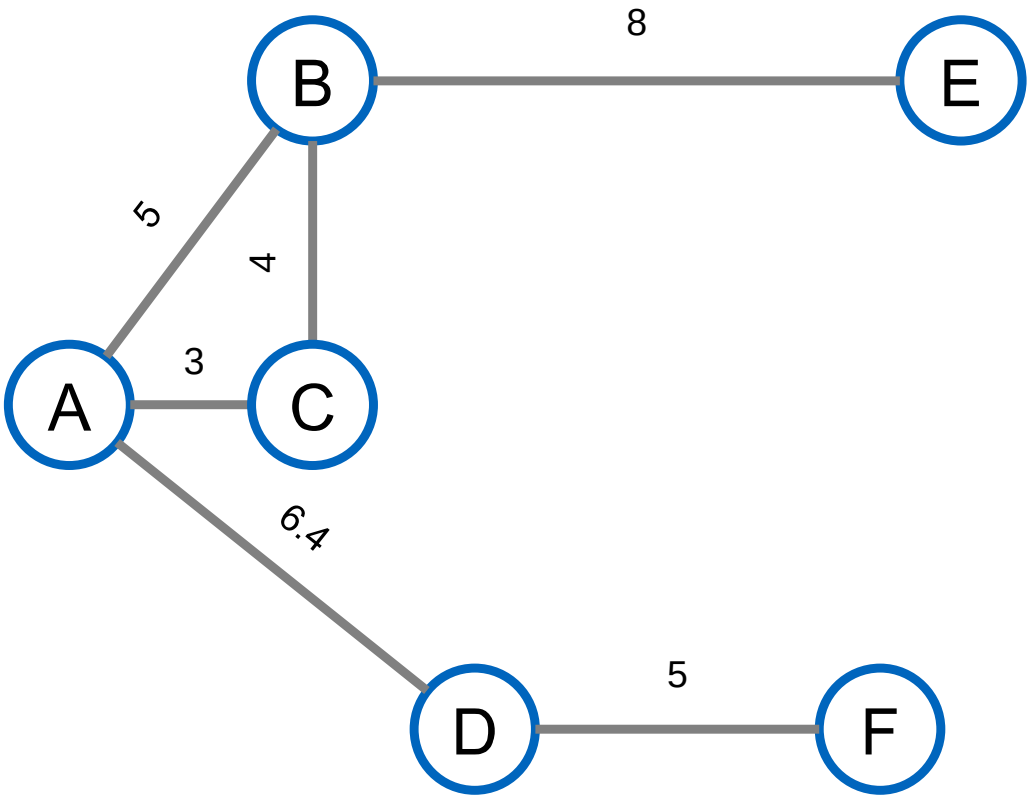
There are two basic groups of complete pathfinding algorithms: Those that guarantee to find a path regardless of cost and those that guarantee to find cost optimal paths. We will take a look at both groups, starting with the group that guarantees to find an arbitrary path connecting origin and destination.

The following algorithms will be presented:

1. Arbitrary, feasible path
   Depth First, Breadth First, Best First
2. Optimal path
   Dijkstra, A*

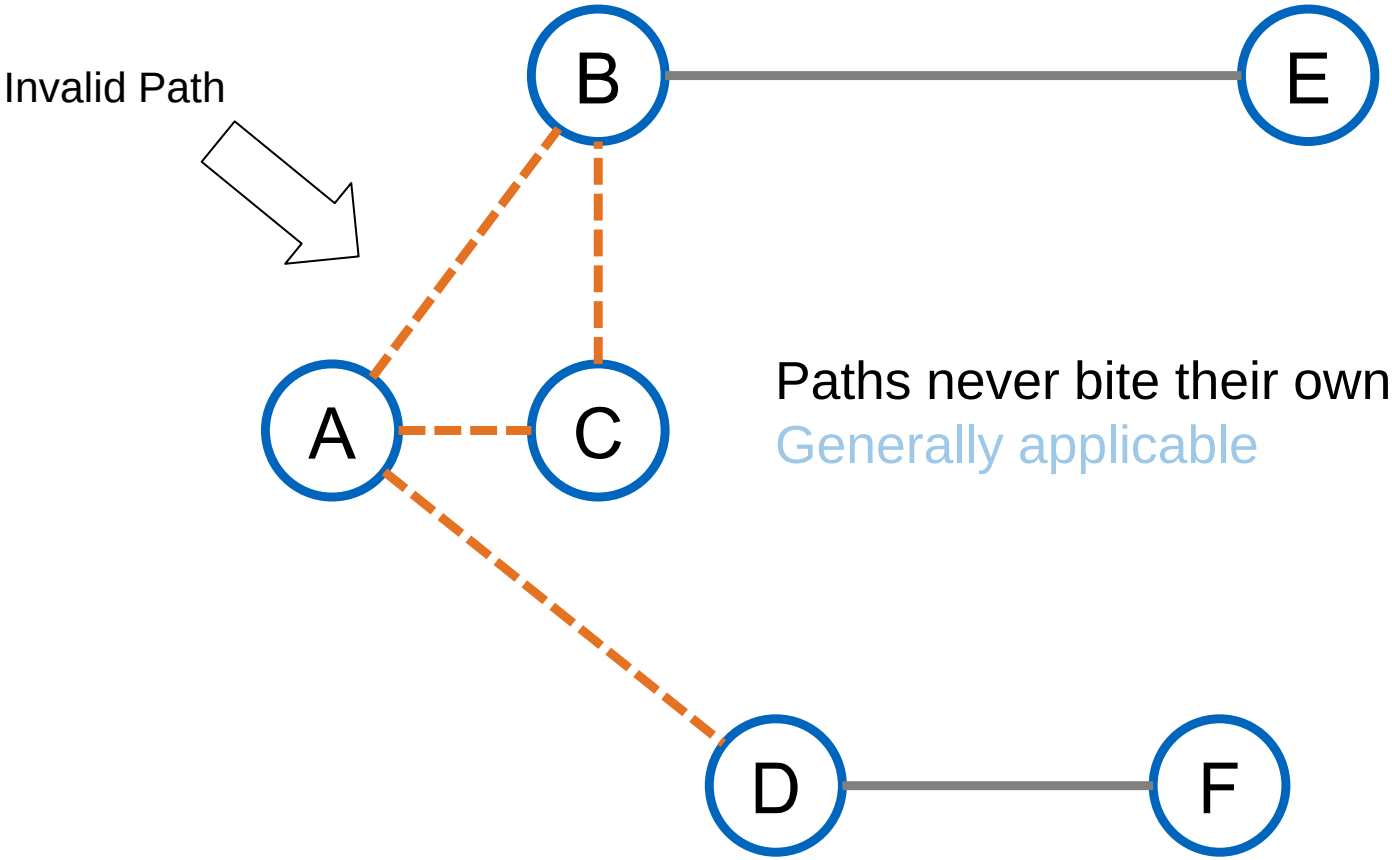
Algorithms

Pathfinding Example Graph



Algorithms

Ground Rules Example Graph

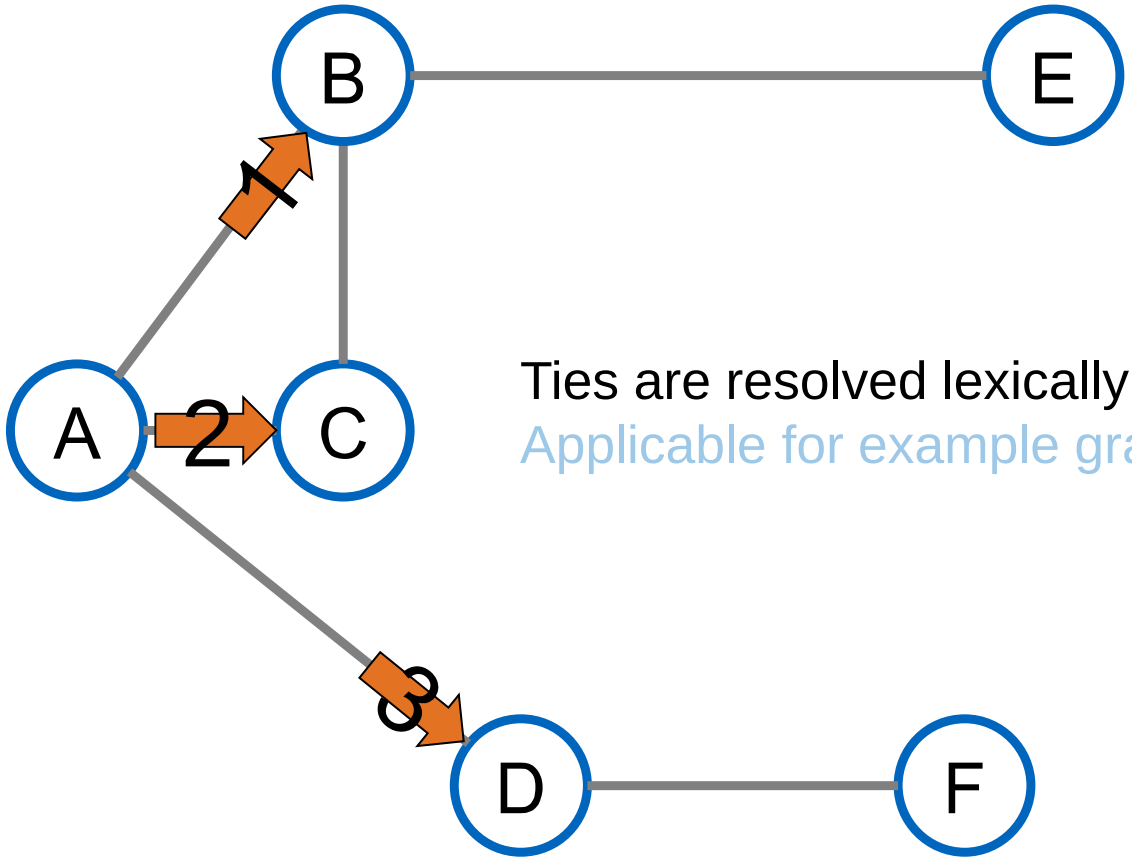


Paths never bite their own tail

Generally applicable

Algorithms

Ground Rules Example Graph



Ties are resolved lexically

Applicable for example graph

Additional Slides

The example graph used in the remainder of this lecture is designed to illustrate the differences among the introduced pathfinding algorithms.

We will use it to iterate through the algorithm steps while visualizing the progress to make the concepts easier and more intuitive to grasp. To do so, the path under investigation will be highlighted by colored edges for each algorithm step.

The example graph comprises six nodes (Labelled A to F) and six edges. The edge weights are given without units but may be understood to be distances between the corresponding nodes. In each example, A is the origin and F is the destination.

During the application of the algorithms, situations will arise, in which nodes that are already on the current path could potentially be visited another time (i.e. the path biting its own tail). This behavior is never desirable for any pathfinding algorithm because it introduces loops that never yield benefits for the path found. In the example, the highlighted partial path with origin A and destination F traverses A twice (A,B,C,A,D), which is always less efficient than a path A, D would be. Therefore, loops in potential paths are always forbidden for any of the presented algorithms. This behavior can easily be implemented by forbidding to add any node to a partial path that is already part of the partial path. This reasoning is generally applicable in real-life applications.

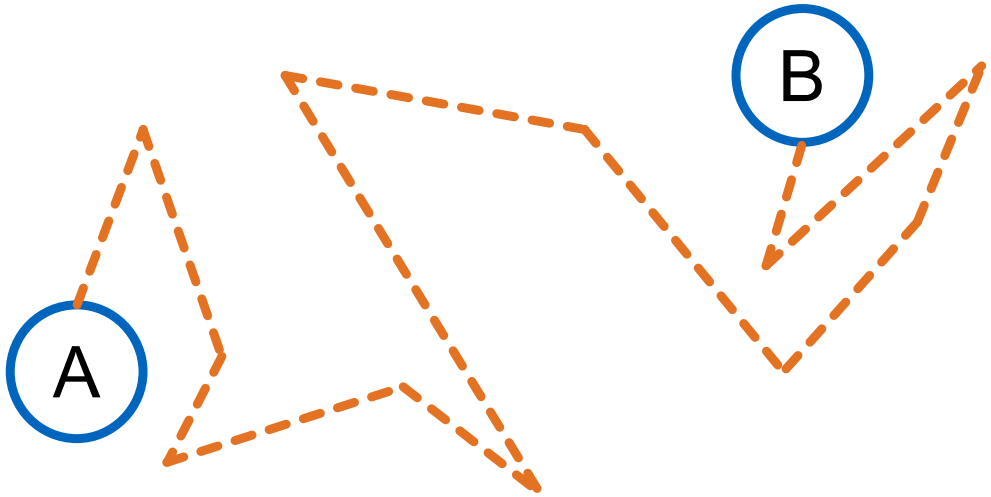
In addition to loops, iterations of a pathfinding algorithm may be confronted with constellations in which two steps are possible. In these situations, we will use lexical order to resolve the tie. This approach is generally not applicable in real-life applications. In these, other measures are taken to resolve ties.

Algorithms – British Museum

Description

Algorithm:

- Randomly append Nodes to path while not stuck.
- If destination was reached: Success
- If stuck, start over



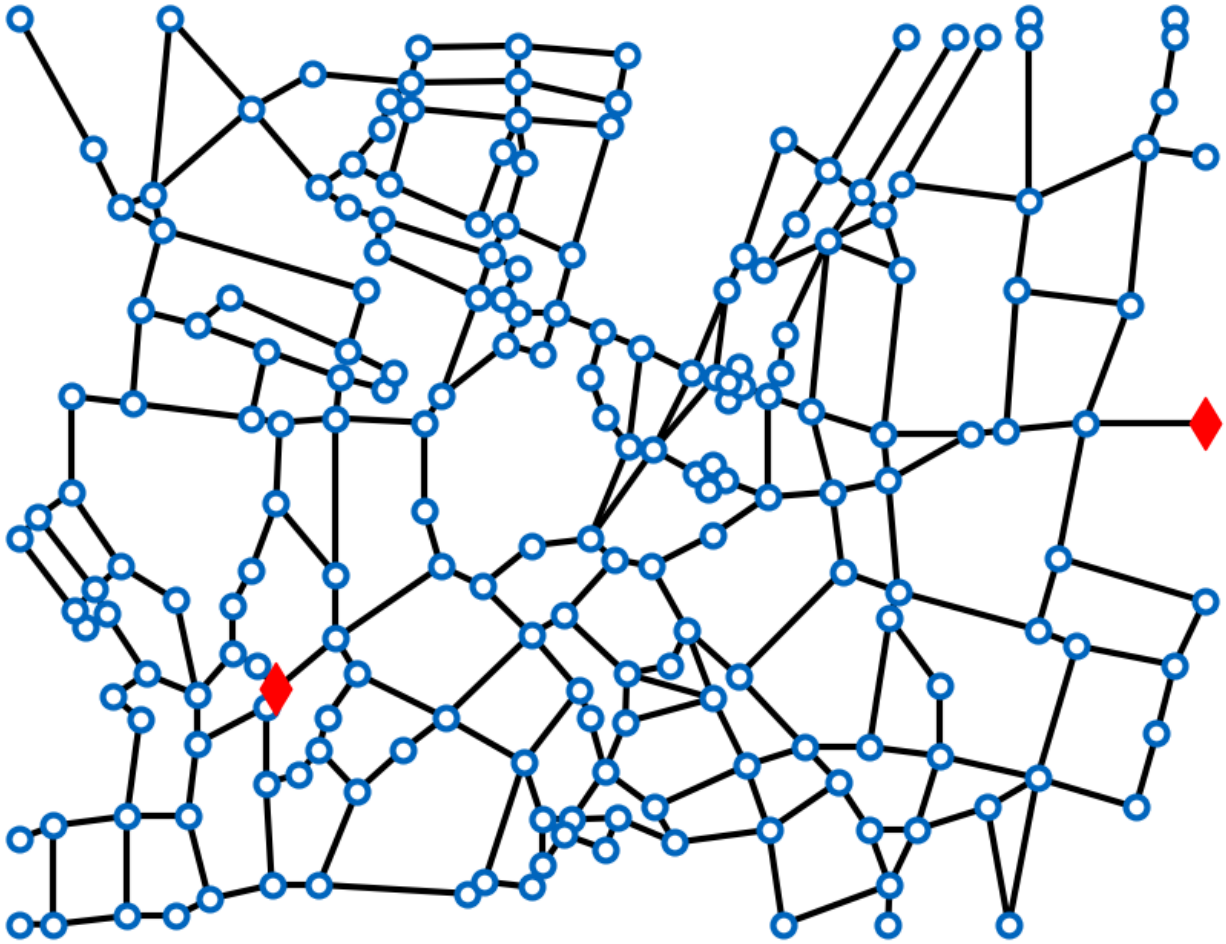
Additional Slides

The structurally most simple way to find a feasible path between origin and destination is known as the “British Museum Approach”. The British Museum approach gets its name from the idea that all books in the British Museum could have been generated by pure randomness. To do so, all possible texts with a maximum word count would have to be generated. Among these would surely be every book text present in the British Museum. The same concept can be transferred to pathfinding. Hence, one can use the British Museum approach to generate random paths starting in the origin, until one is found that terminates in the destination rather than in a dead end.

Obviously, this approach is a very chaotic approach that is most inefficient due to massive repetitions of ineffective sub-paths. Furthermore, it does not employ any reasoning about the paths generated. Nevertheless, it is complete.

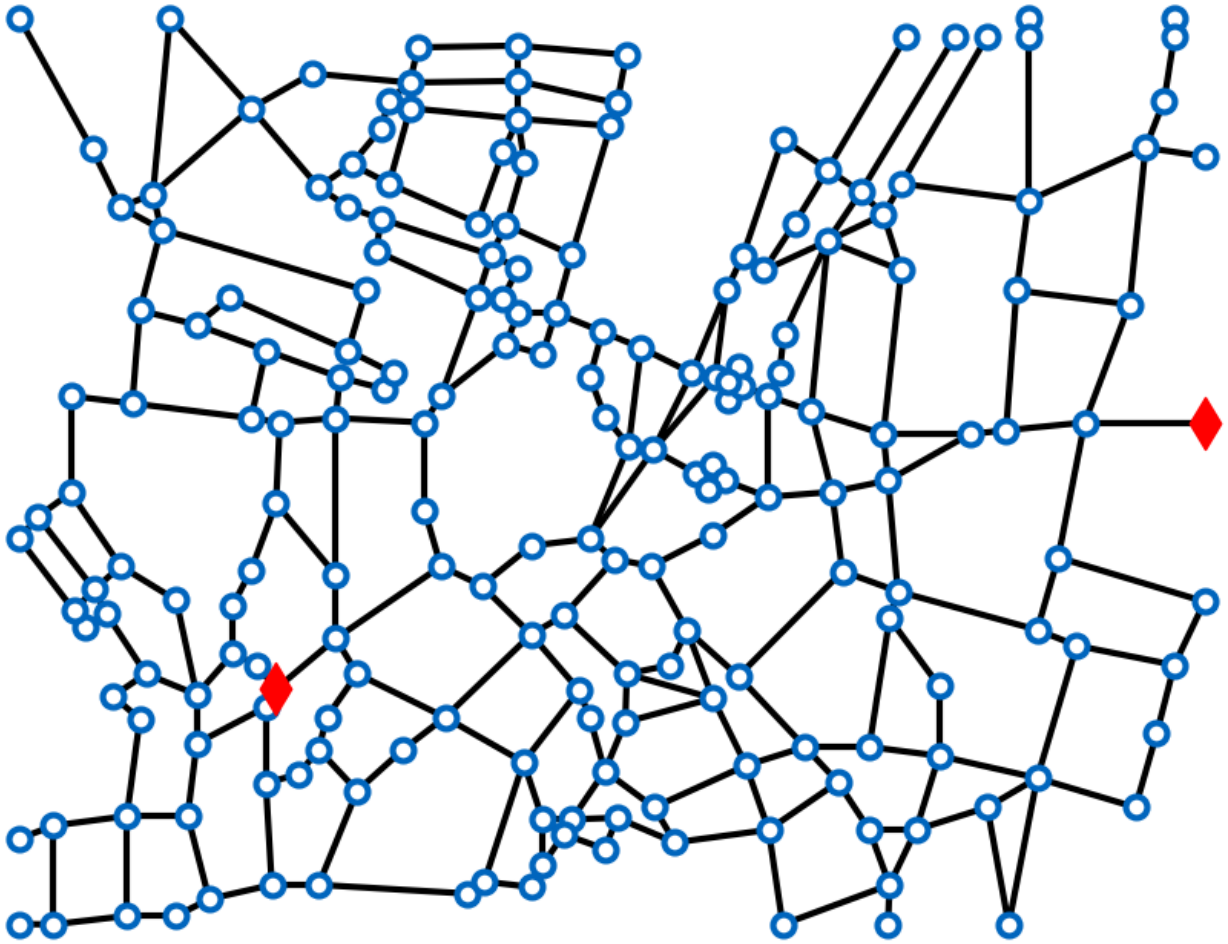
Algorithms

Munich Demo Graph



Algorithms – British Museum

Munich Demo



Additional Slides

The animation shows how the British Museum approach successfully determines a path between origin and destination in an example graph comprising the main roads of the Munich street network. It is evident that there is no gradual convergence of the algorithm.

It is by pure chance that one of the paths generated by the British Museum approach is feasible. The resulting path is – in this case – a feasible, but very unintuitive path between origin and destination. It contains many unnecessary turns and does hardly prefer the general direction of the destination, although it passes many edges that clearly follow a more direct path toward the destination.

Furthermore, the algorithm generates a huge number of paths that absolutely diverge from the destination by starting off in the wrong direction in the first place.

The British Museum algorithm generates a lot of paths before a feasible path is found. It is very inefficient, although it could potentially find an optimal path on the first try. For larger networks, the British Museum algorithm is not usable, because it explores massive proportions of the network and provides no structural reasoning that guides the search.

In real-life approaches, where the network is basically not constrained but large and highly connected, the British Museum algorithm might generate excessive paths before searching the direct surroundings of the origin when searching for the destination. E.g. a navigation system using such an approach might suggest you to follow a path through Italy when going from Garching Forschungszentrum to Marienplatz.

To overcome the downsides of the British Museum approach, structured approaches are used to reduce the number of iterations needed and to guarantee a deterministic, time bound convergence towards the destination.

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

---

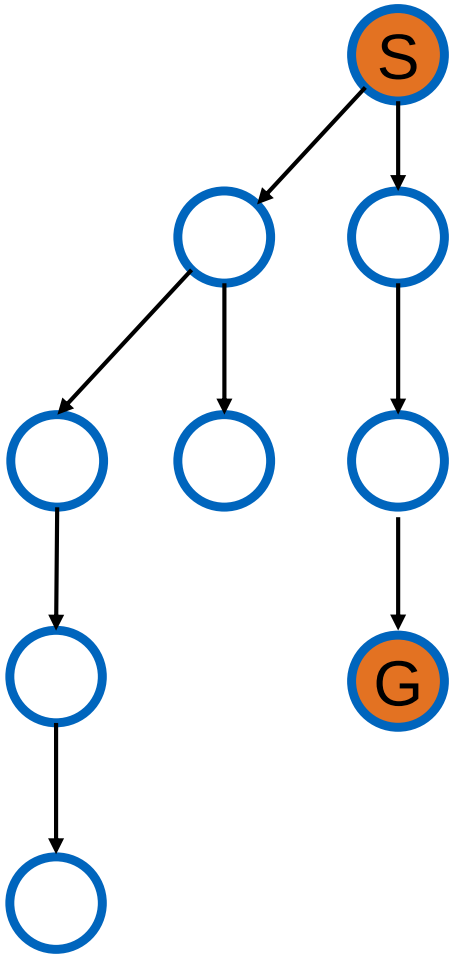
- Until the first path in the queue is empty,
  - Remove the first path from the queue, create new paths by extending the first path to all neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the front of the queue

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

Diving into the Search Tree

Algorithms – Depth First

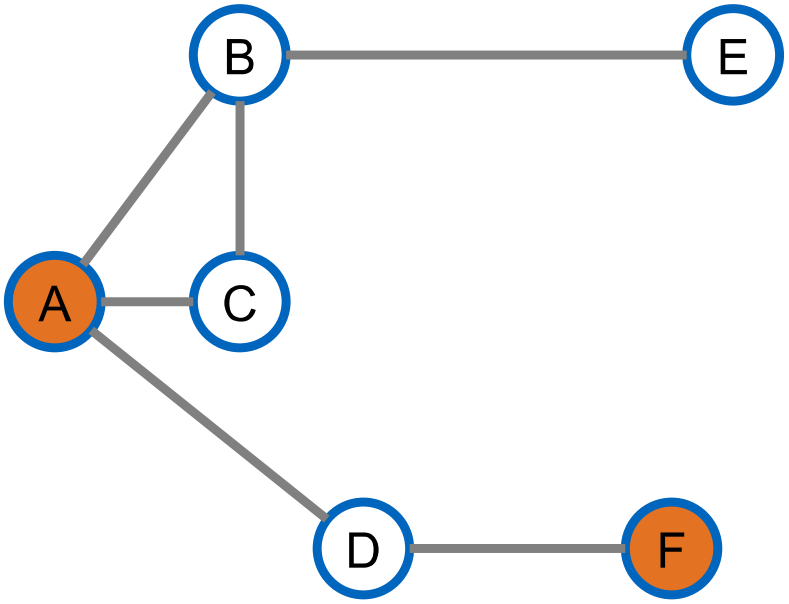
Characteristic Search Tree



narrow and deep

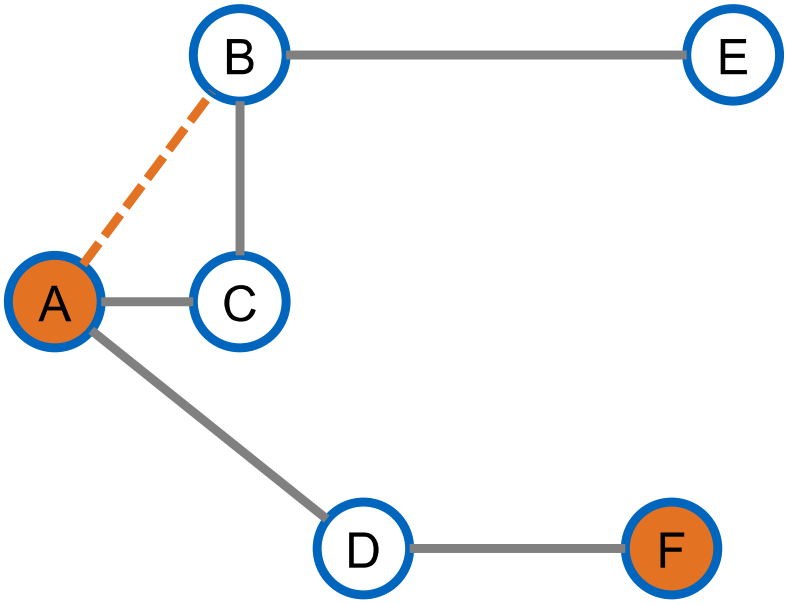
Algorithms – Depth First

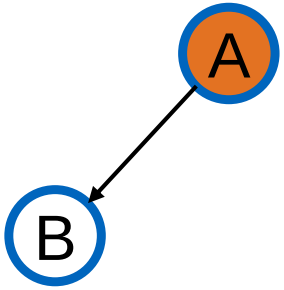
Pathfinding Example Graph



Algorithms – Depth First

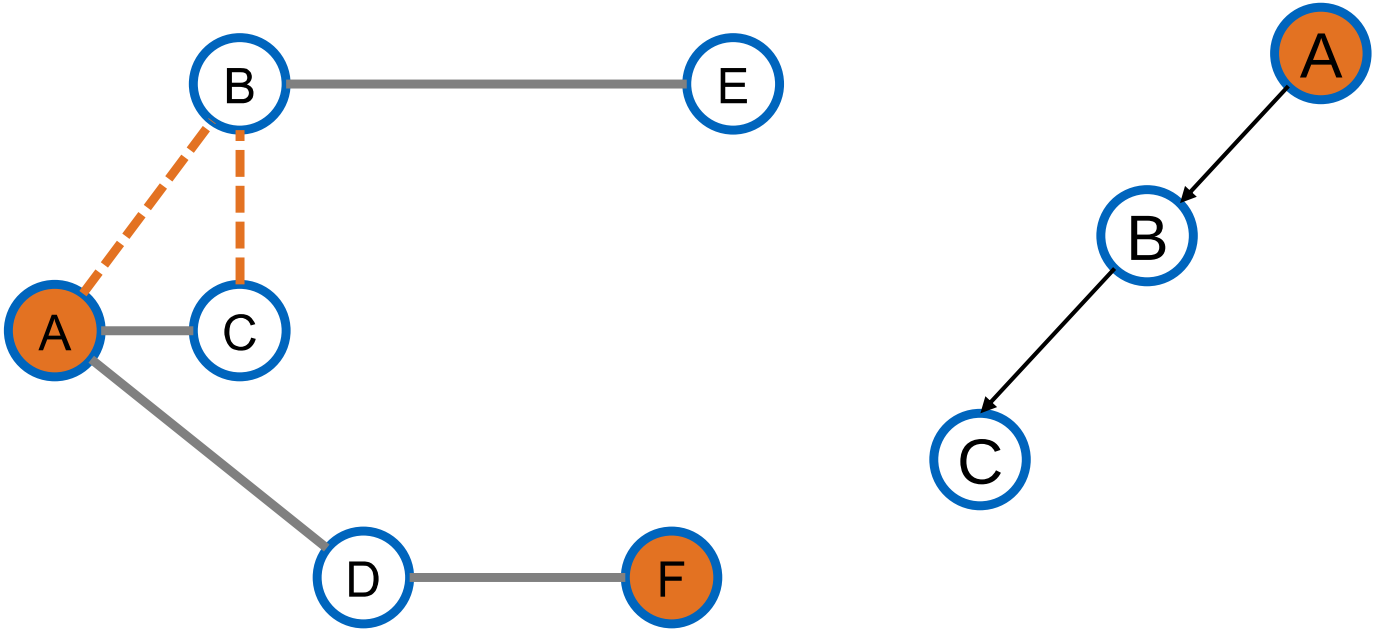
Pathfinding Example Graph





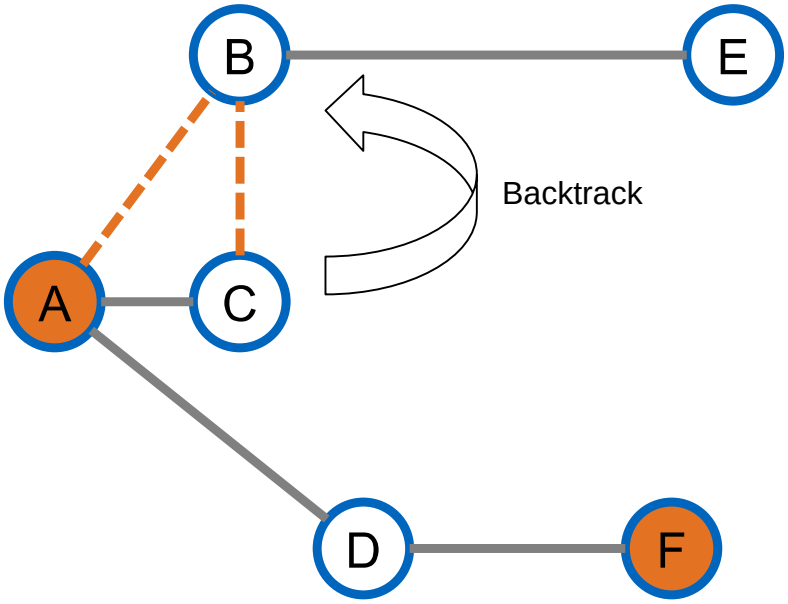
Algorithms – Depth First

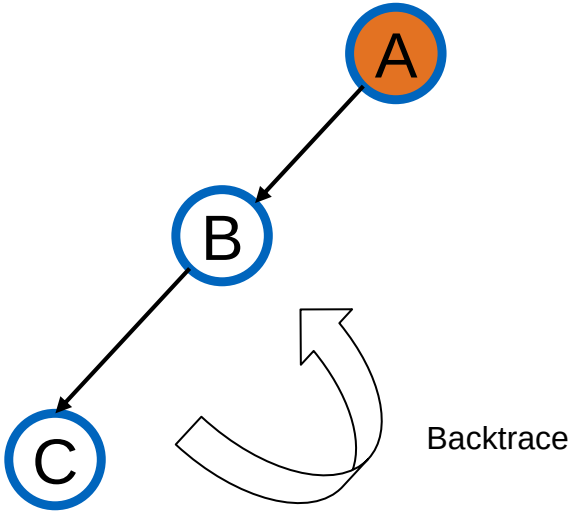
Pathfinding Example Graph



Algorithms – Depth First

Pathfinding Example Graph



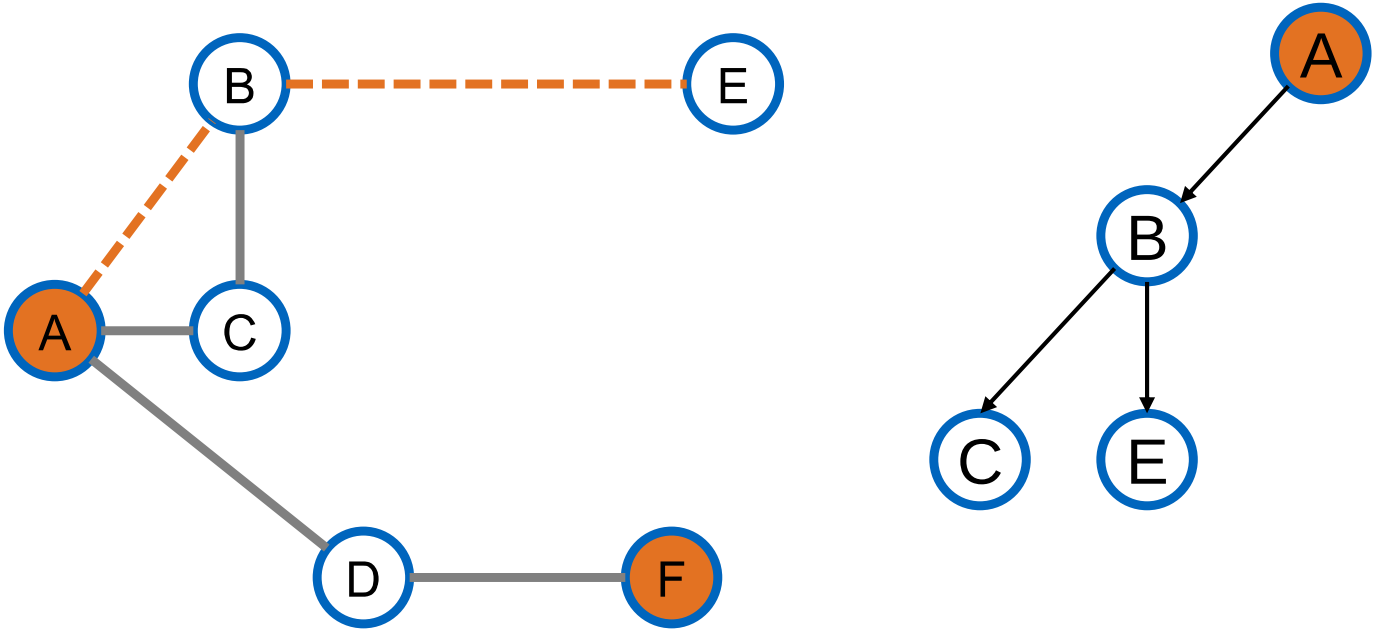


Additional Slides

During the application of the Depth First algorithm, situations may arise in which the current path terminates in a node from which on no node is reachable that is not yet on the current path. In these situations – as well as in dead ends – Depth Frist employs a mechanism called “backtracking” or “backtracing”. Backtracking makes the algorithm jump back to the last point were a decision can be made and carries on from that point, until all alternative at that node are exhausted. If that is the case and no feasible path has been found, the algorithm jumps back to the crossroads before et cetera. This behavior can be sustained until all possible paths starting at the origin have been tried. If the algorithm still fails to find a path, no solution is possible in the graph, i.e. there is no connection between the origin and destination in the graph.

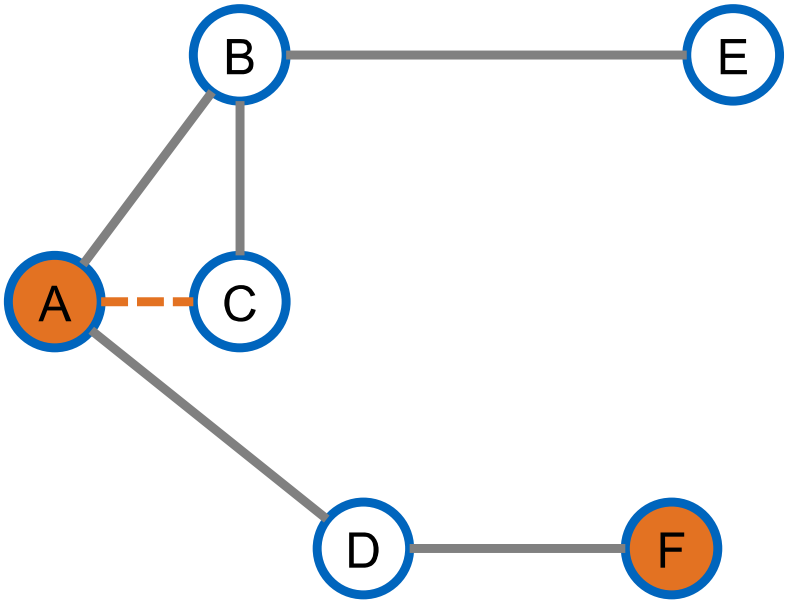
Algorithms – Depth First

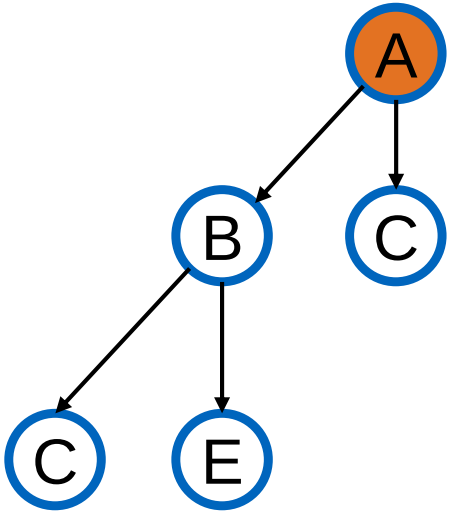
Pathfinding Example Graph



Algorithms – Depth First

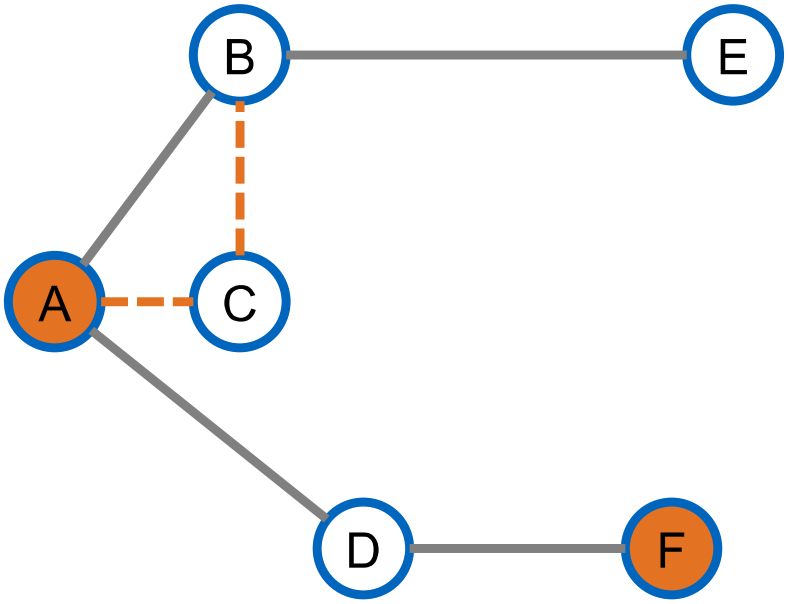
Pathfinding Example Graph

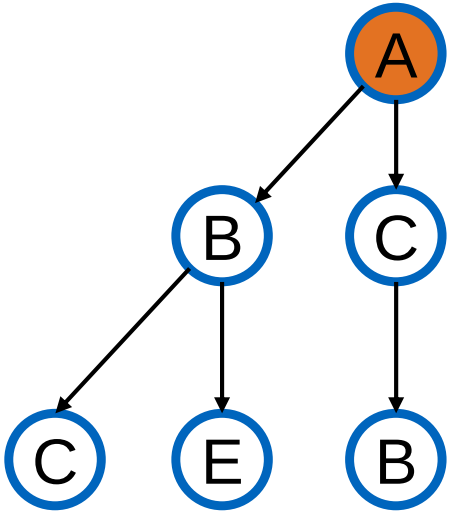




Algorithms – Depth First

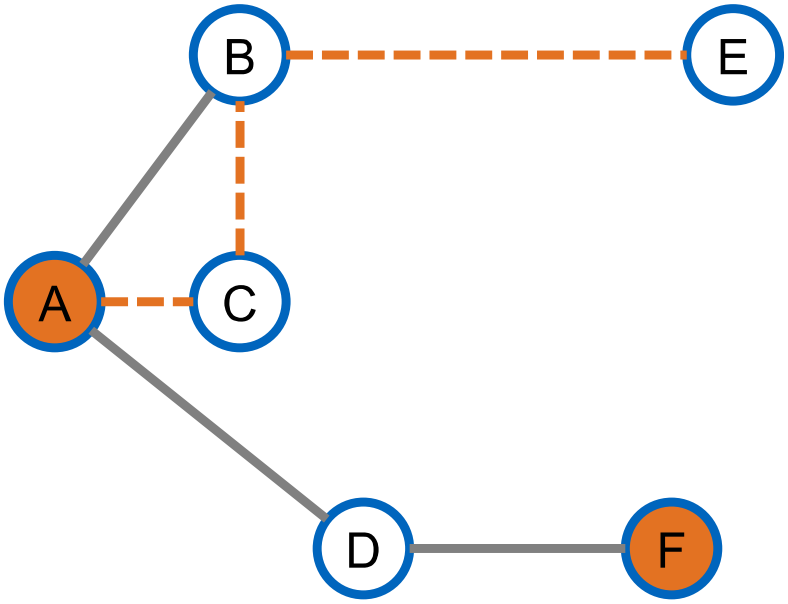
Pathfinding Example Graph

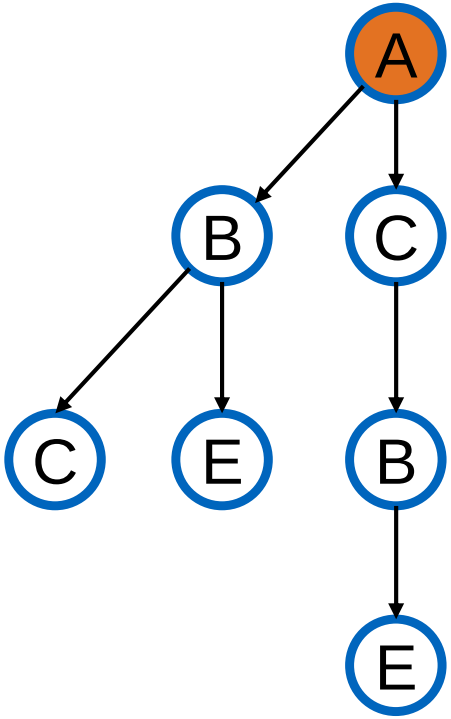




Algorithms – Depth First

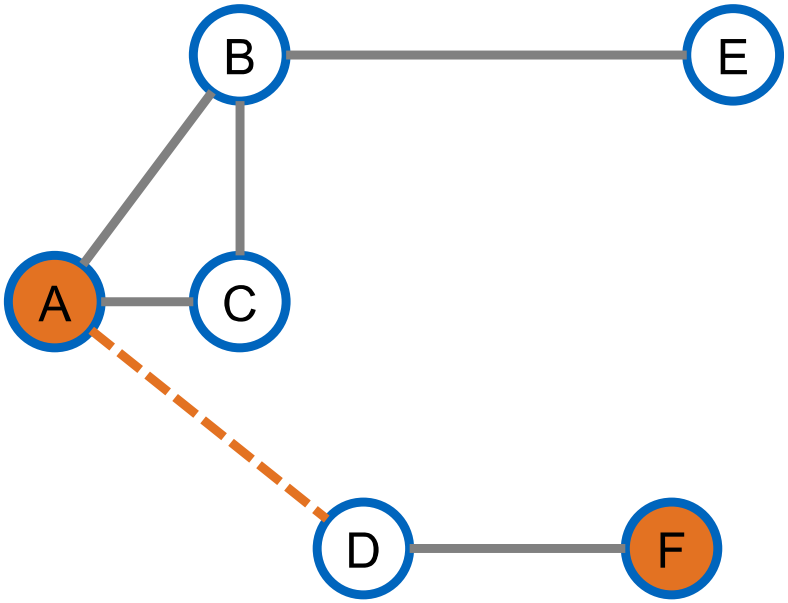
Pathfinding Example Graph

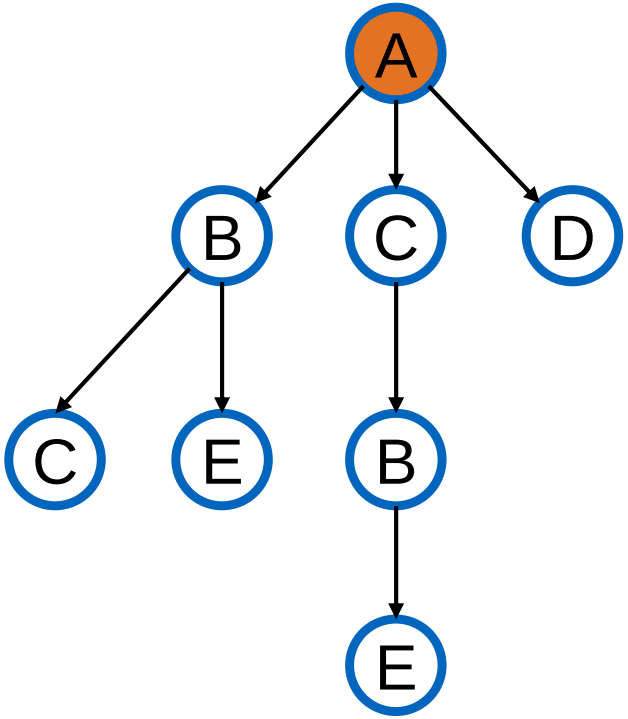




Algorithms – Depth First

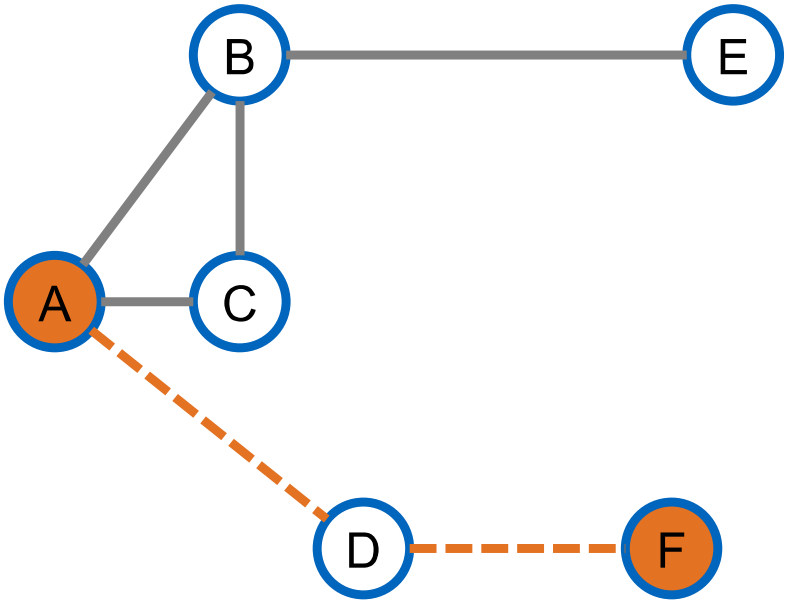
Pathfinding Example Graph

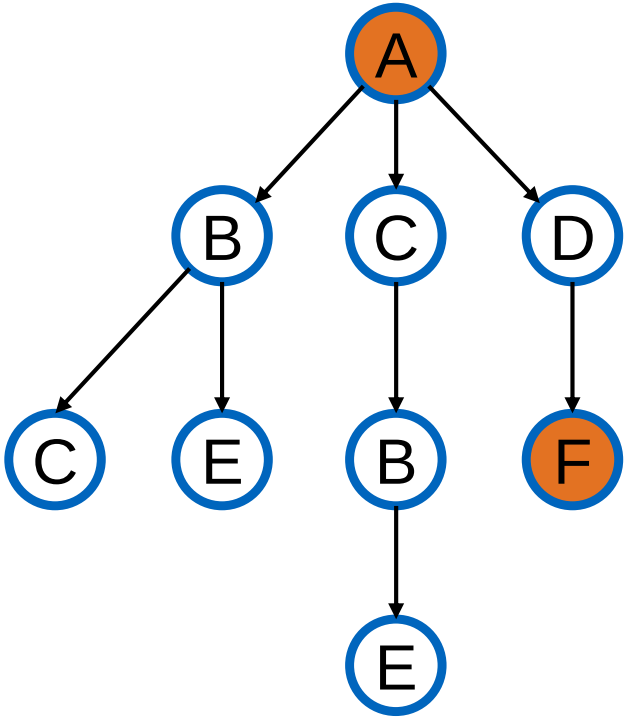




Algorithms – Depth First

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

---

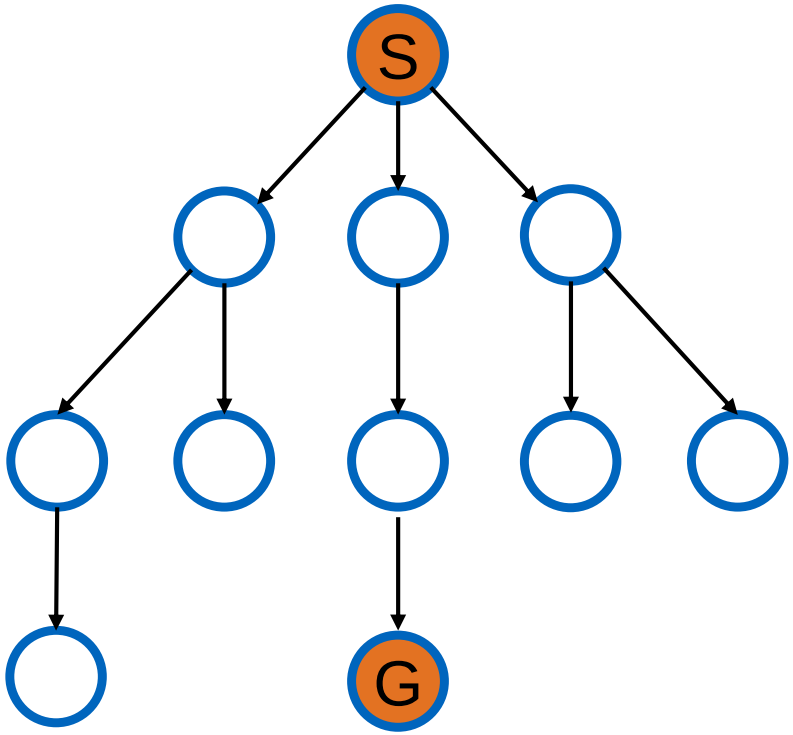
- Until the first path in the queue ends at the goal node or the queue is empty,
  - Remove the first path from the queue, create new paths by extending the first path to the neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the back of the queue

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

Scanning the Search Tree Level by Level

Algorithms – Breadth First

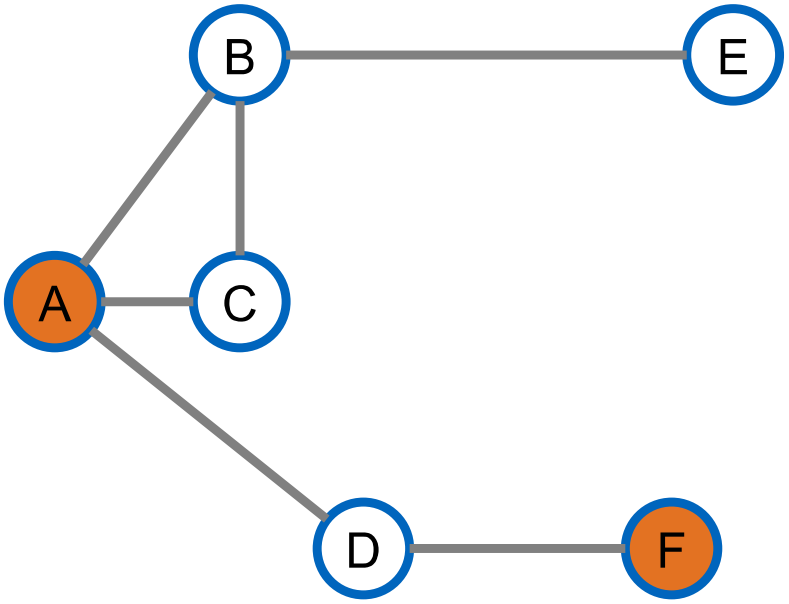
Characteristic Search Tree



wide and shallow

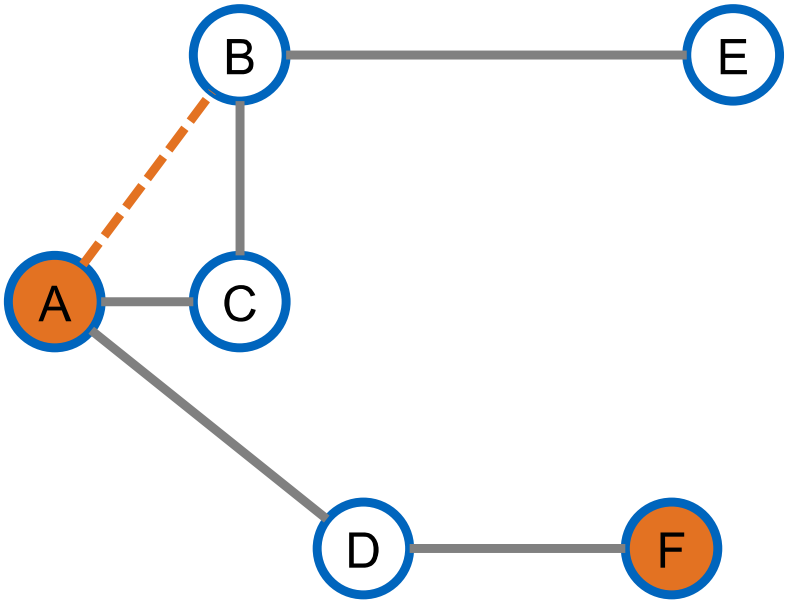
Algorithms – Breadth First

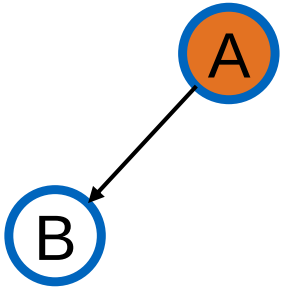
Pathfinding Example Graph



Algorithms – Breadth First

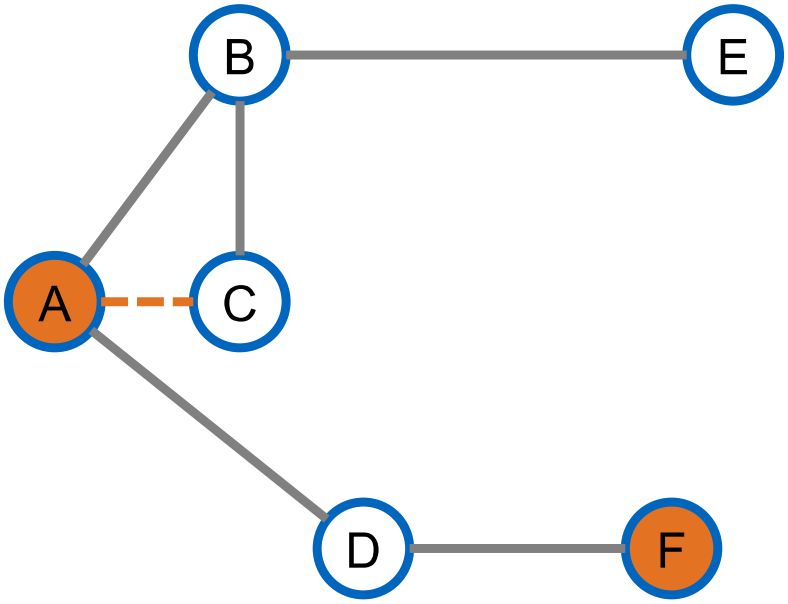
Pathfinding Example Graph

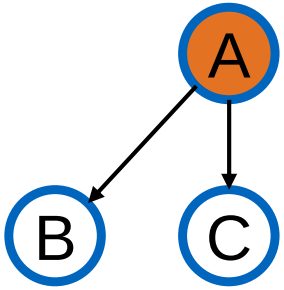




Algorithms – Breadth First

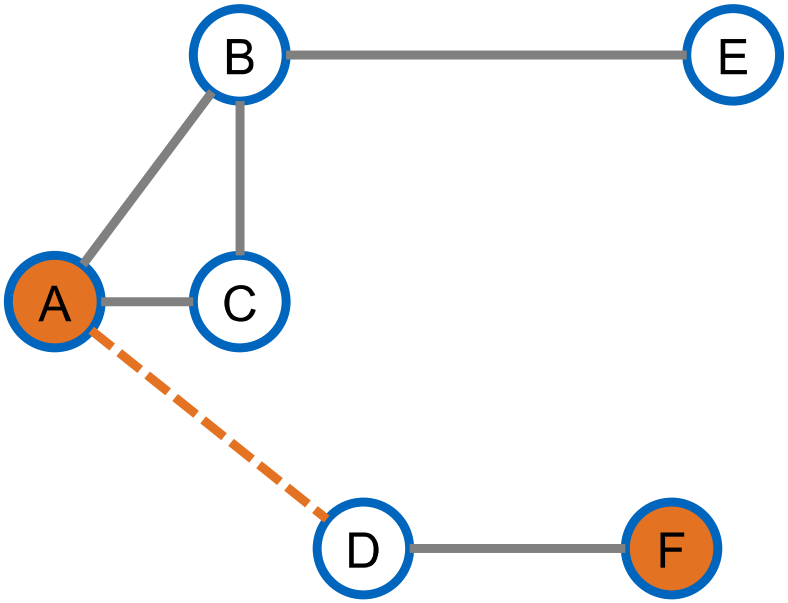
Pathfinding Example Graph

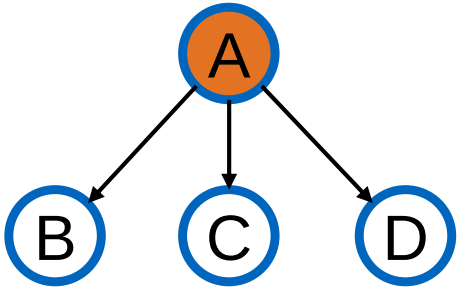




Algorithms – Breadth First

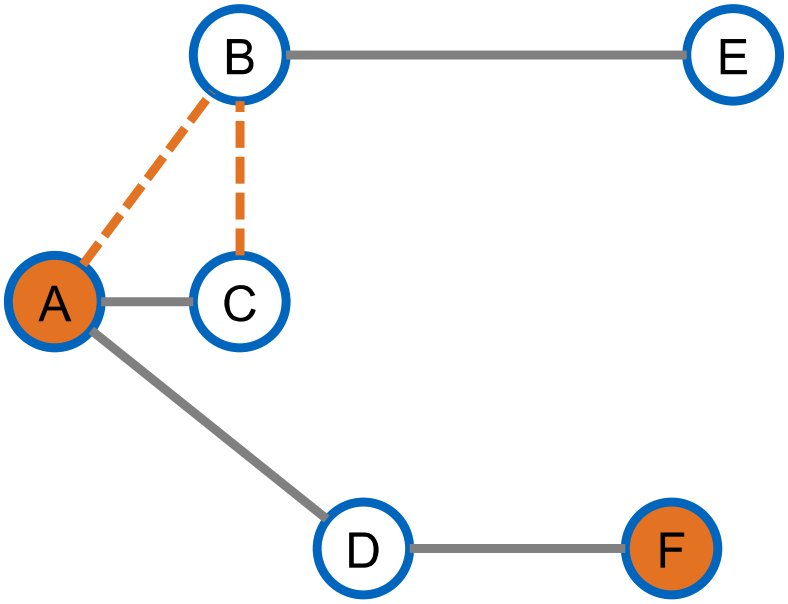
Pathfinding Example Graph

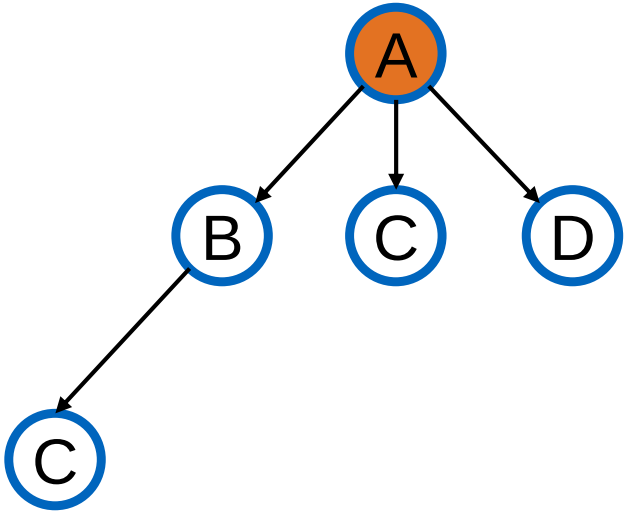




Algorithms – Breadth First

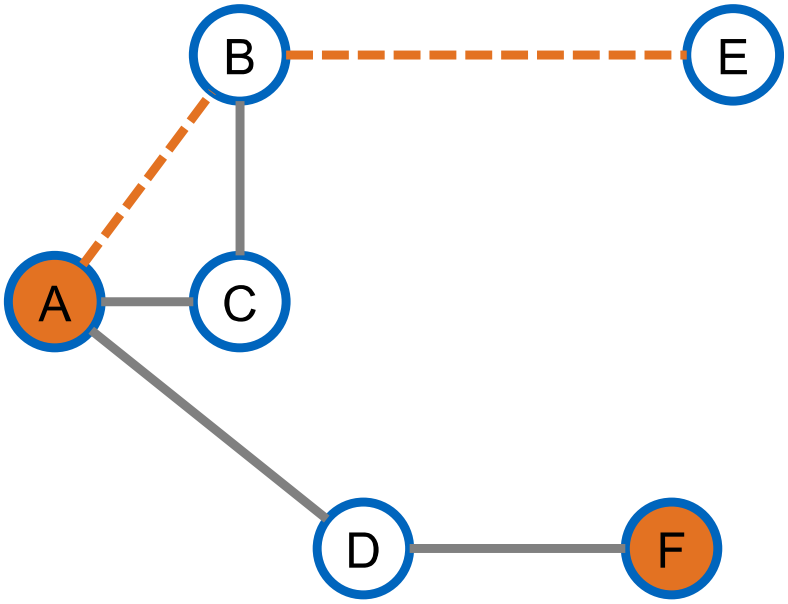
Pathfinding Example Graph

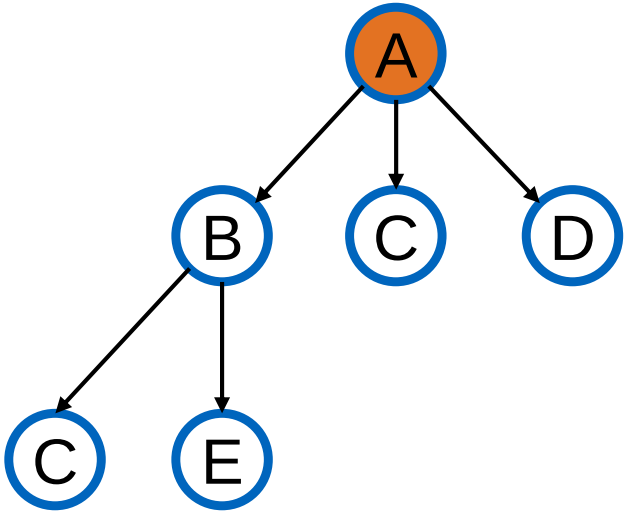




Algorithms – Breadth First

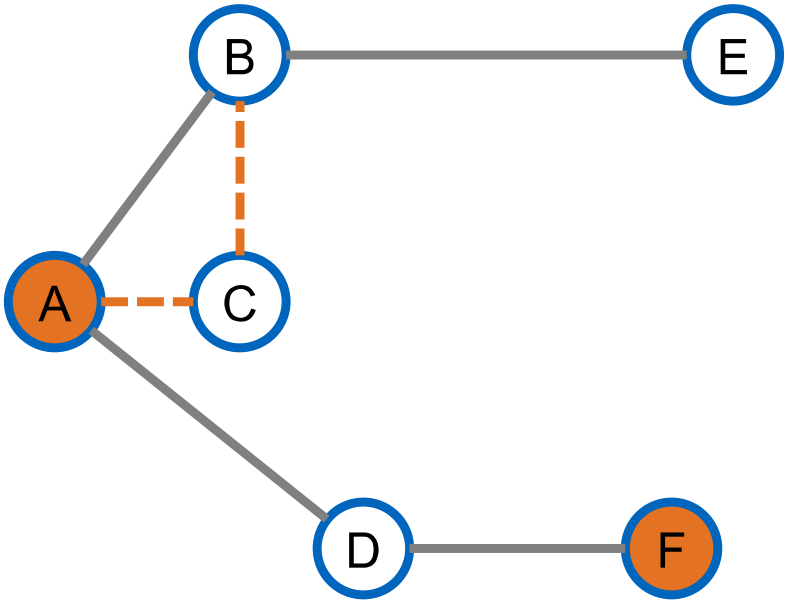
Pathfinding Example Graph

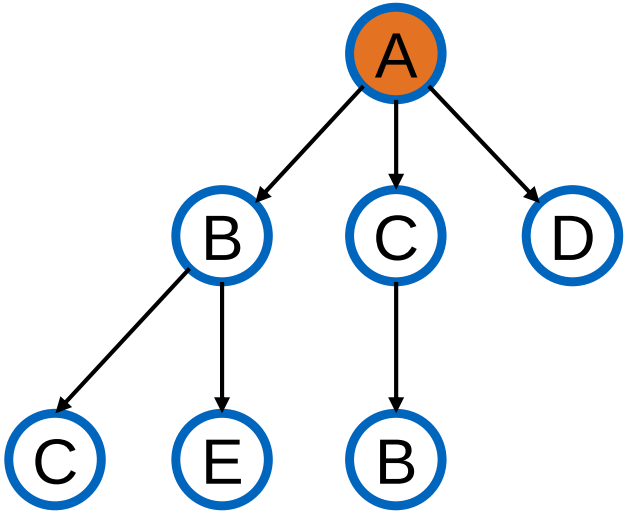




Algorithms – Breadth First

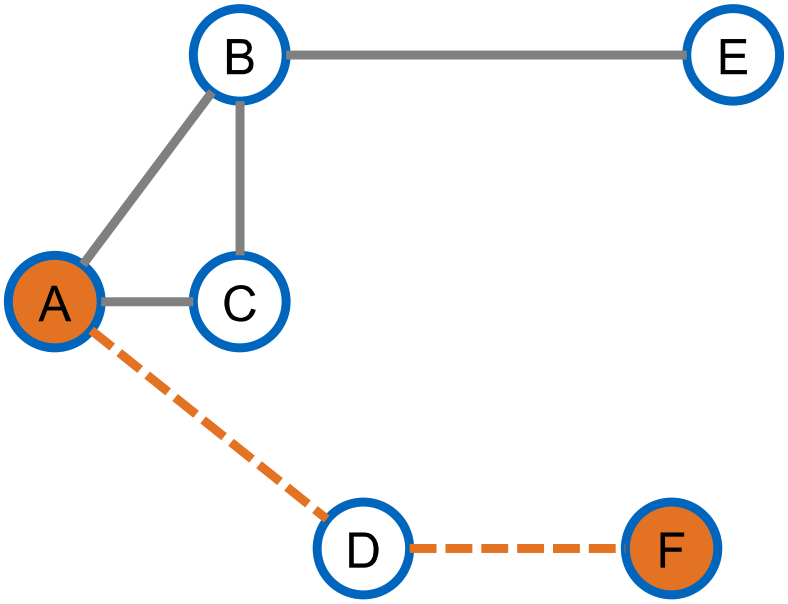
Pathfinding Example Graph

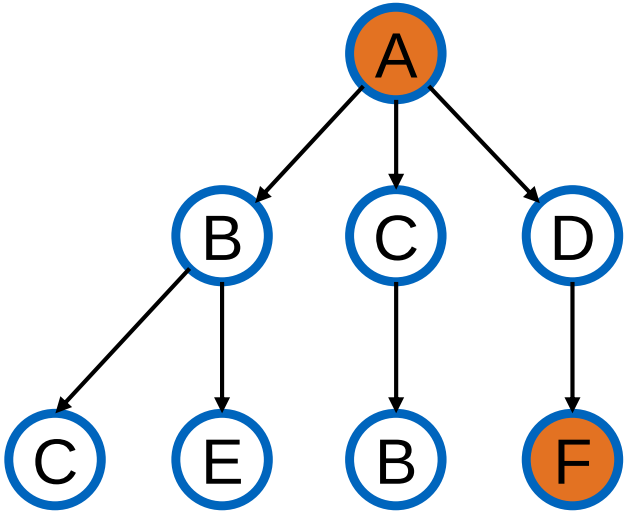




Algorithms – Breadth First

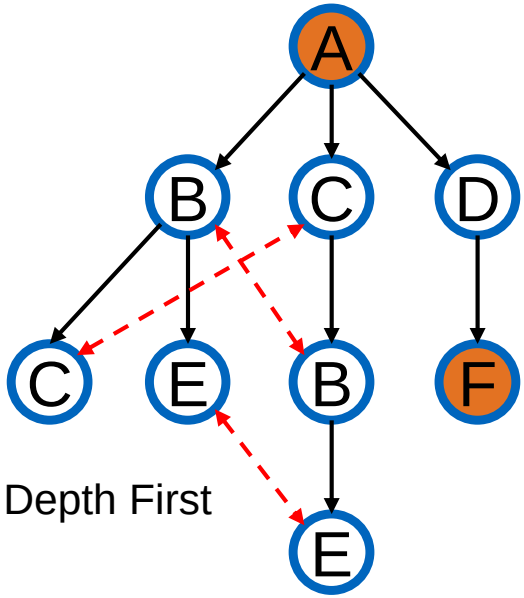
Pathfinding Example Graph



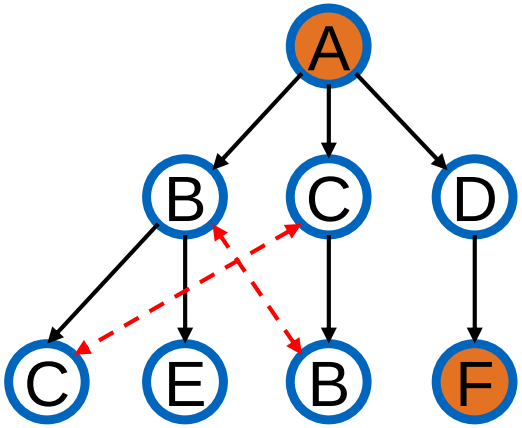


Algorithms – Enhancements

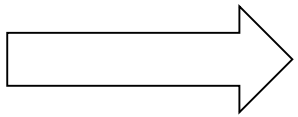
Shortcoming of Basic Algorithms - Revisitation



Depth First



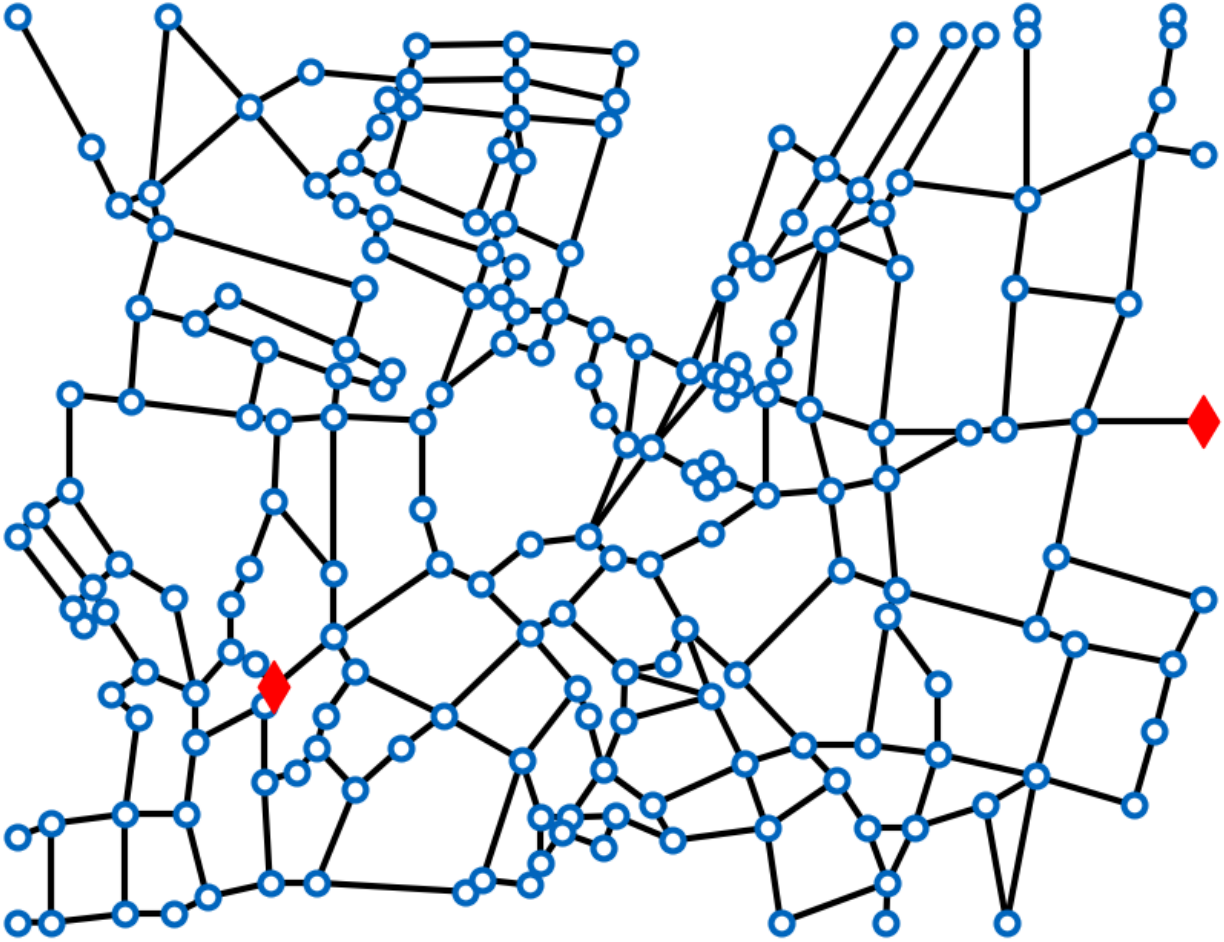
Breadth First



Prevent Revisitation

Algorithms – Breadth First Enhanced

Munich Demo



Additional Slides

The animation shows an enhanced breadth first algorithm on the street network of Munich. In contrast to the British Museum approach, it jumps around a lot, trying out different partial paths, before adding nodes to the partial path. Through this behavior, it evolves around the origin in a structured manner, resembling a circle, until it converges toward the goal. Just like the British Museum approach, the Breadth First algorithm fails to follow the general direction of the target, but refrains from generating excessive paths before searching the surroundings of the origin.

If you consider the resulting path, you will find that there is no possible path between origin and destination that crosses less nodes. This is a general property of the Breadth First algorithm, because it scans the search tree level by level, before appending more nodes to the path candidates.

Algorithms – Enhancements
Informedness

A search algorithm is informed if it employs additional information about the problem that has the potential of guiding the search toward its goal.

Algorithms – Best First

Description

Algorithm:

- 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 unvisited neighbors of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the queue
  - Sort all paths by an heuristic function evaluated at their terminal nodes

---

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

Algorithms – Best First

Description

Algorithm:

- 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 [illegible] the queue is empty,
  - Remove [illegible] paths by extending the first path [illegible] of the terminal node
  - Reject all new paths with loops
  - Add the new paths, if any, to the queue
  - Sort all paths by an heuristic function evaluated at their terminal nodes

---

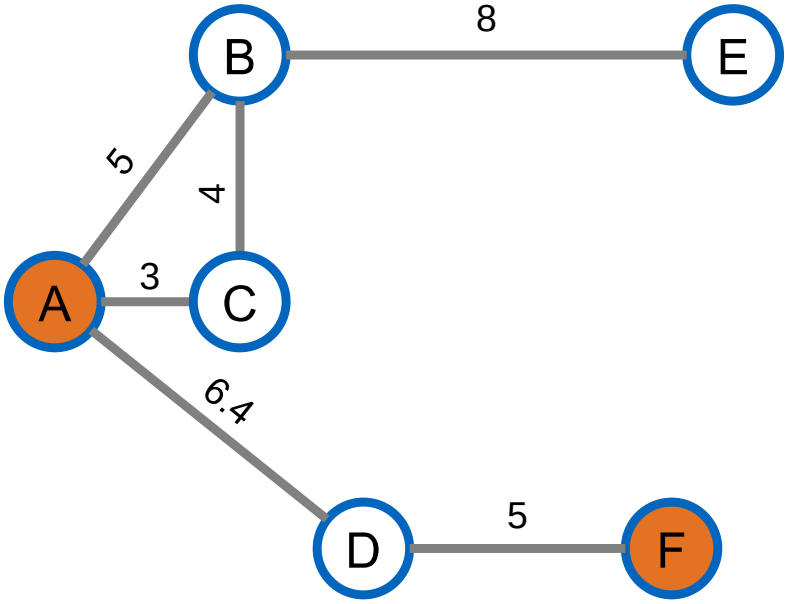
- If the goal node is found, announce success; otherwise announce failure

Always Take the Seemingly Best Step

Algorithms – Best First

Pathfinding Example Graph

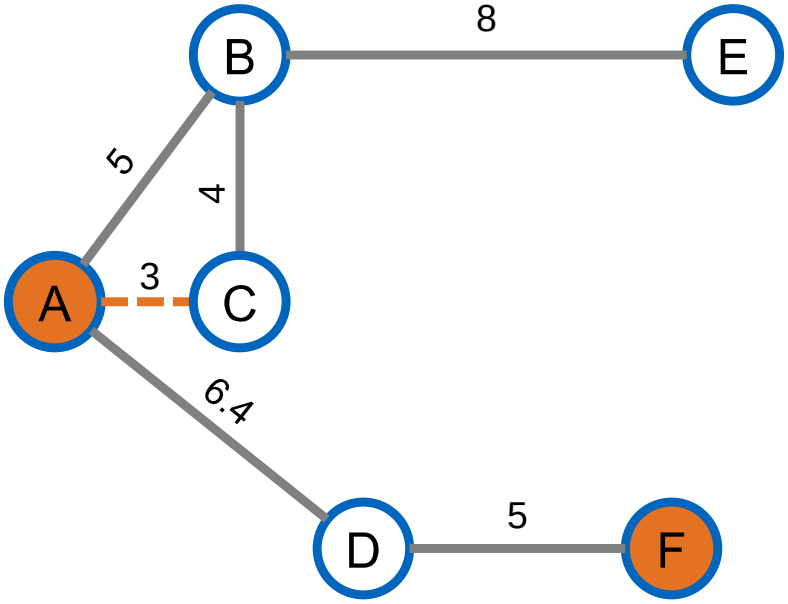
Example: Shortest Step Heuristic

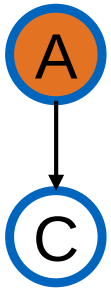


Algorithms – Best First

Pathfinding Example Graph

Example: Shortest Step Heuristic

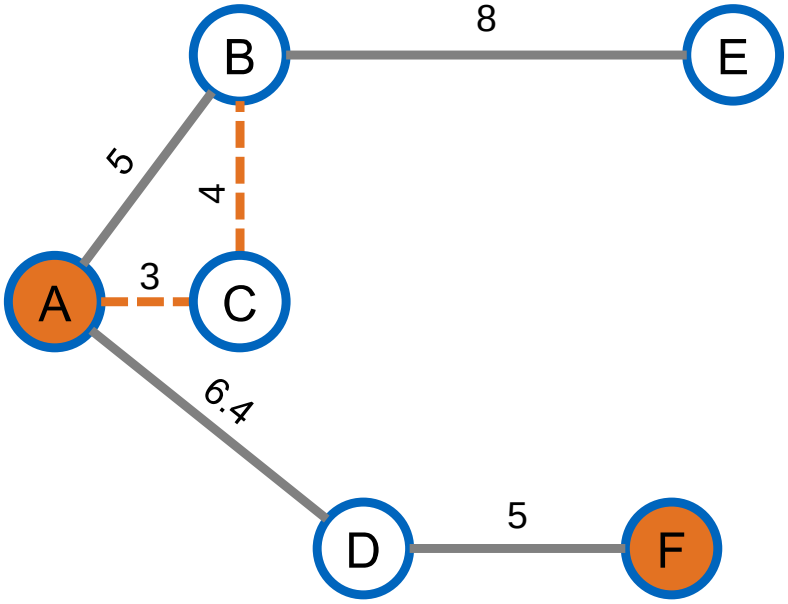


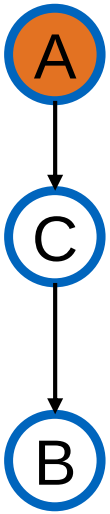


Algorithms – Best First

Pathfinding Example Graph

Example: Shortest Step Heuristic

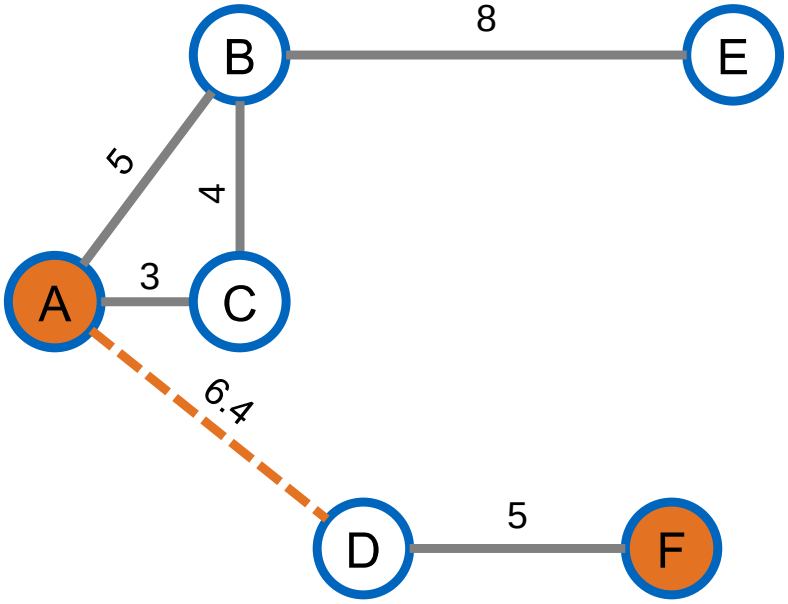


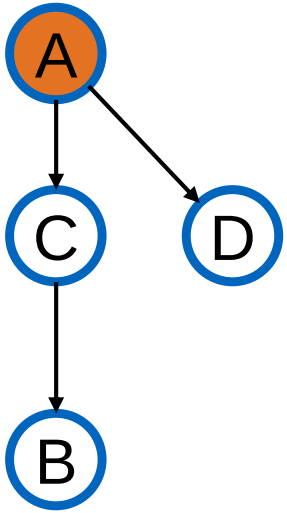


Algorithms – Best First

Pathfinding Example Graph

Example: Shortest Step Heuristic

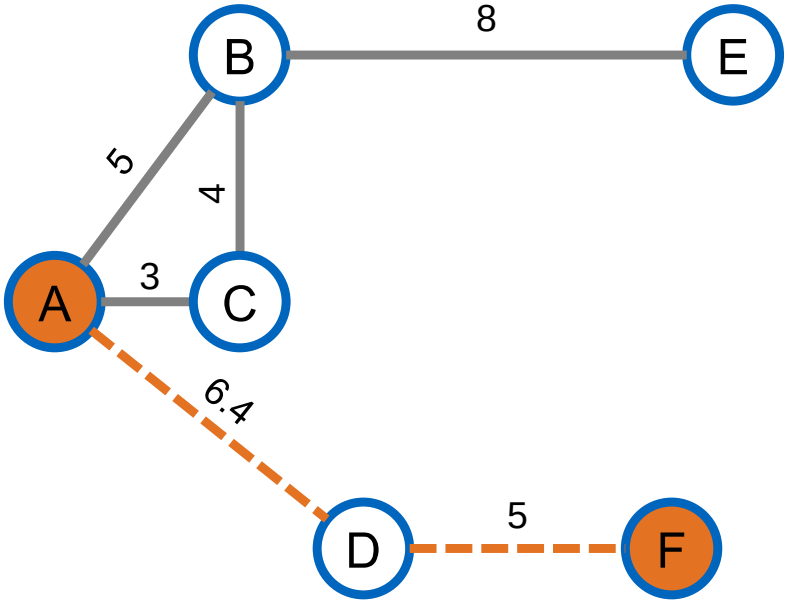


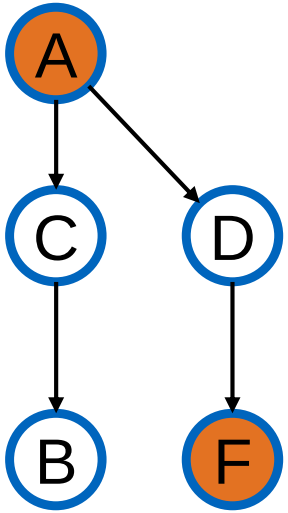


Algorithms – Best First

Pathfinding Example Graph

Example: Shortest Step Heuristic



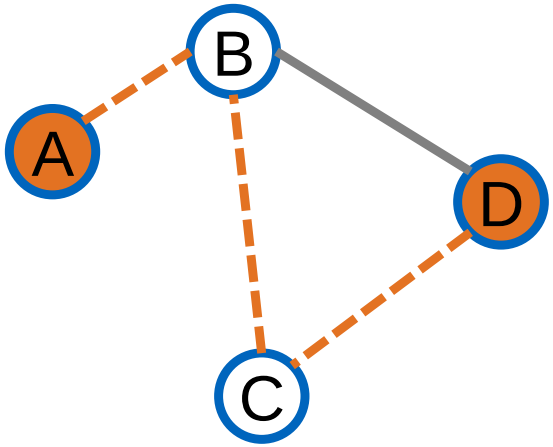


Algorithms

Pathfinding: Problem Formulations

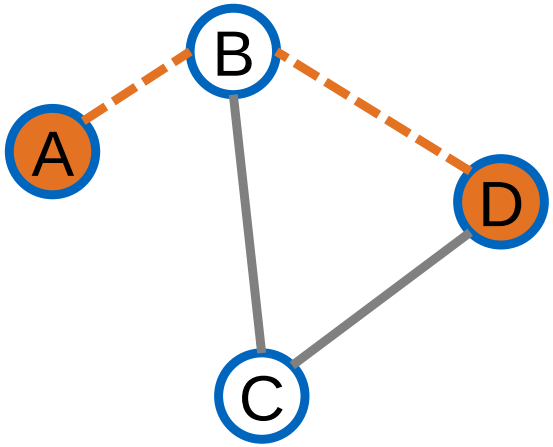
A : Find a Path

- Depth First
- Breadth First
- Best First



B : Find an optimal Path

- Dijkstra
- A*



Algorithms – 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

Algorithms – 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

Algorithms – Dijkstra

Description

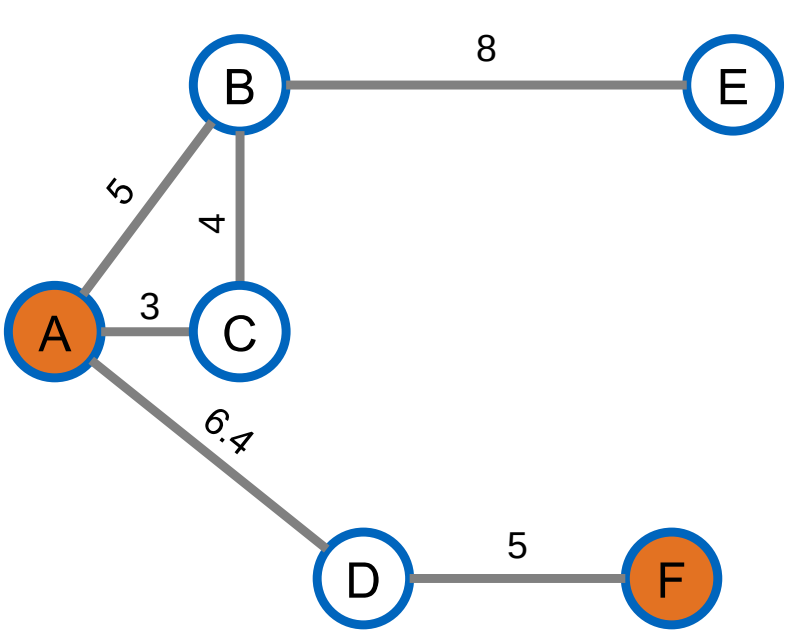
Algorithm – Core Idea:

- 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 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

Explore all Shortest Paths until Goal is Reached

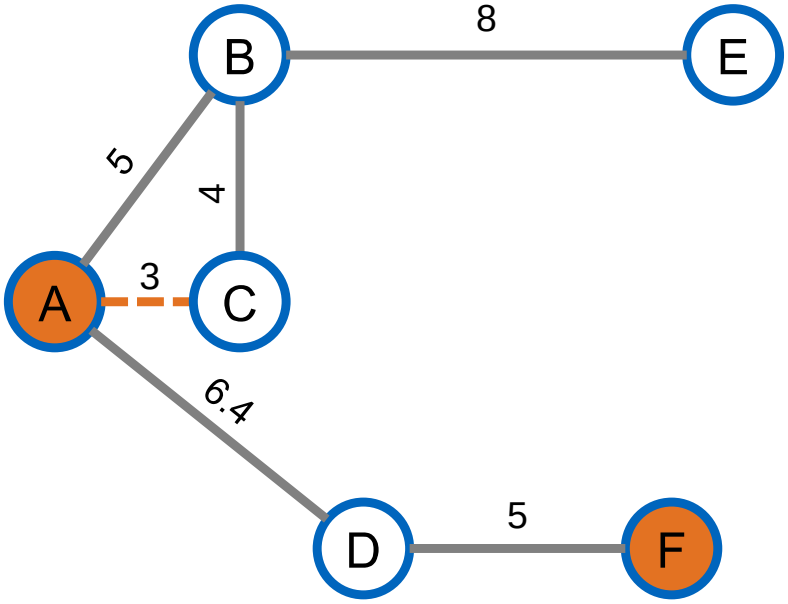
Algorithms – Dijkstra

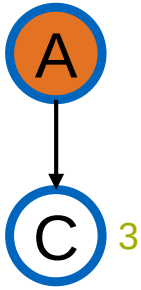
Pathfinding Example Graph



Algorithms – Dijkstra

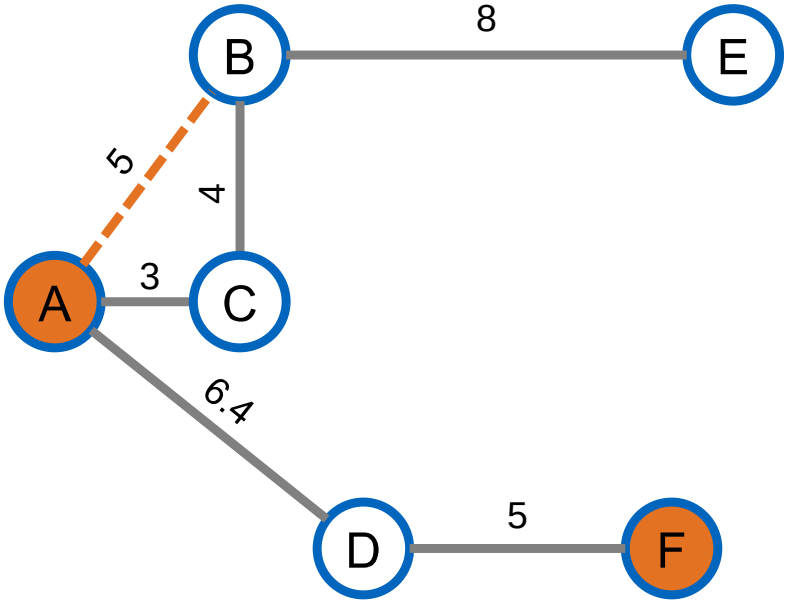
Pathfinding Example Graph

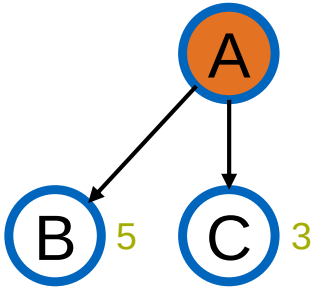




Algorithms – Dijkstra

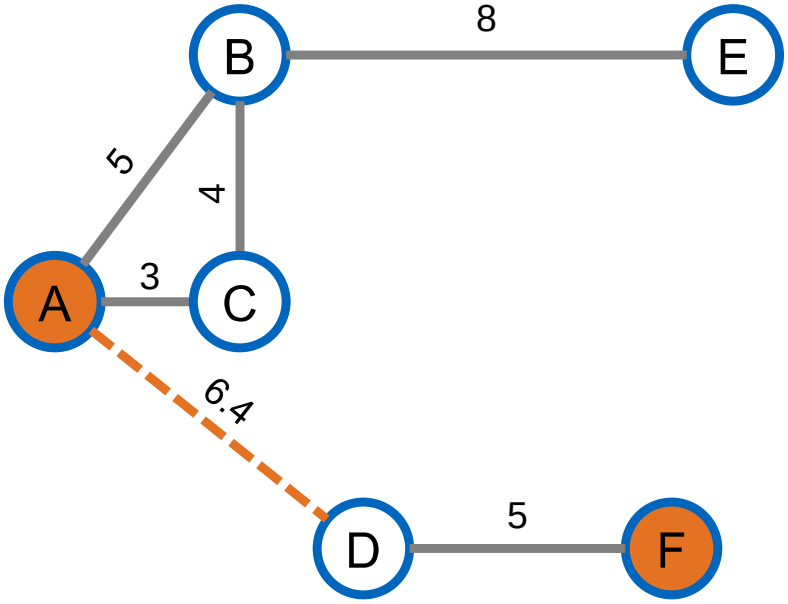
Pathfinding Example Graph

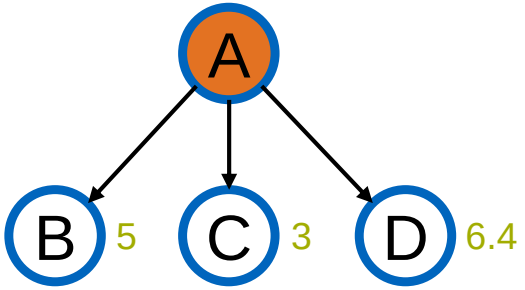




Algorithms – Dijkstra

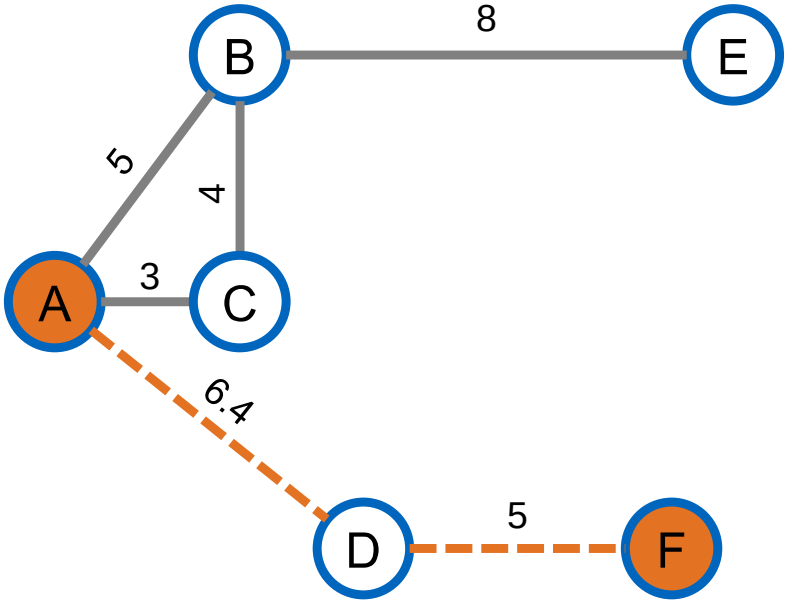
Pathfinding Example Graph

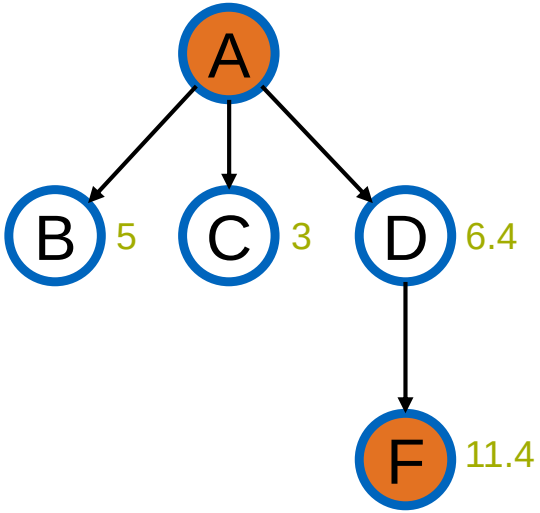




Algorithms – Dijkstra

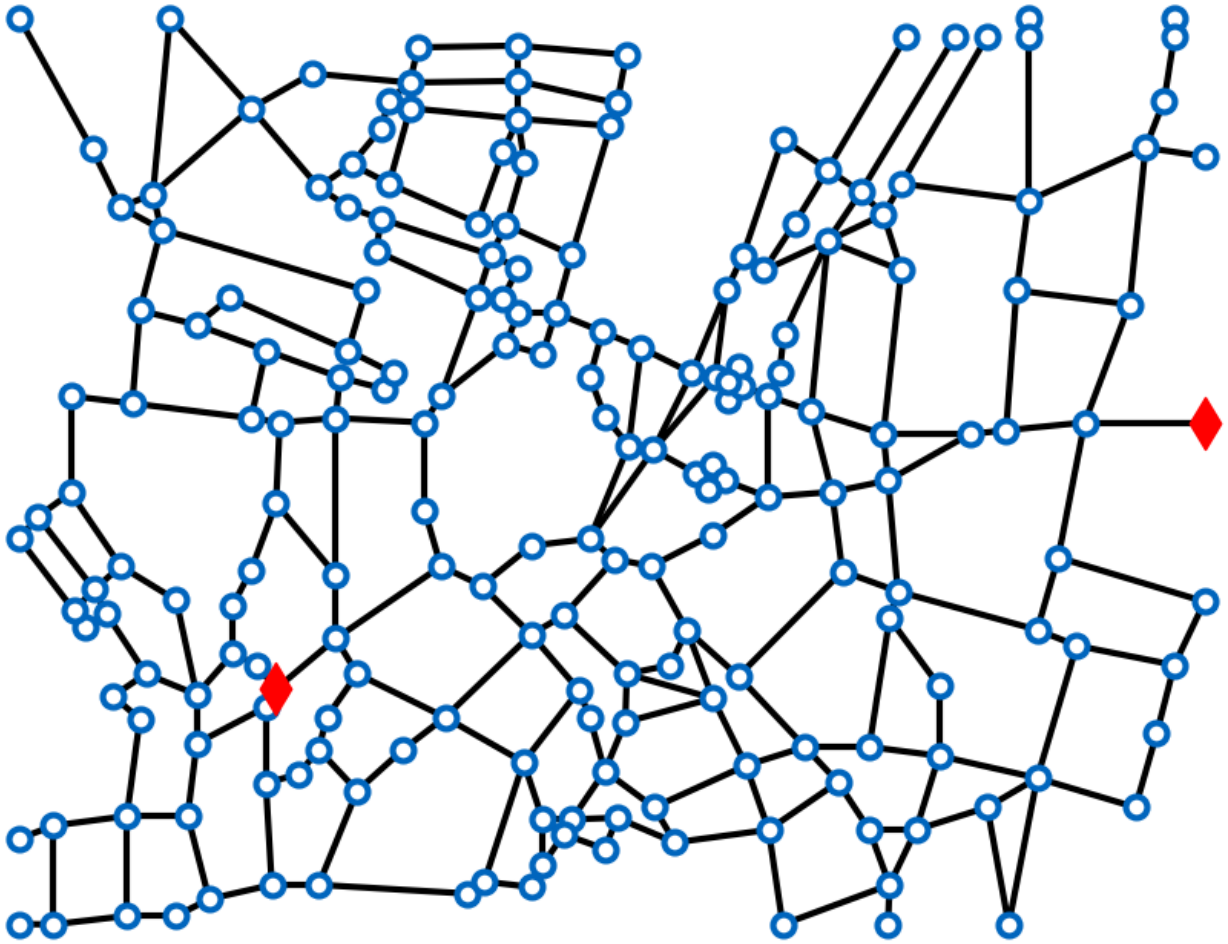
Pathfinding Example Graph





Algorithms – Dijkstra

Munich Demo



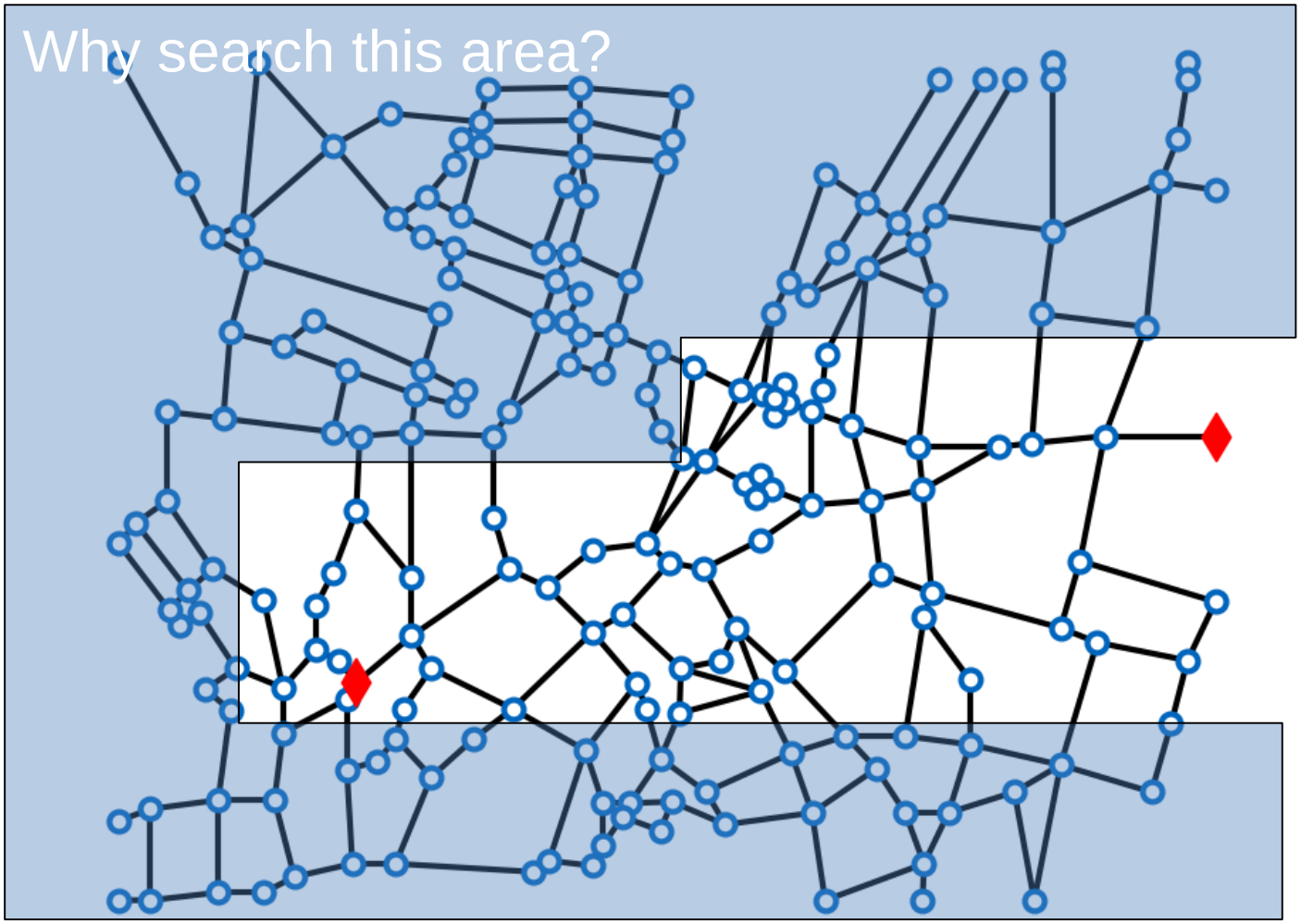
Additional Slides

Much like the Breadth Frist algorithm, the Dijkstra algorithm explores the street network in a circular fashion, generating the most cost efficient paths until one of them terminates in the goal node. Therefore, it is not only guaranteed to find a feasible path, like Breadth First, but also to find a cost optimal path. However, it does not employ any information about the distance to the goal. Hence, all paths that are shorter than or equally short to the optimal path have to be examined regardless of their potential to reach the destination with optimum cost. I.e. a partial path that starts off in the wrong cardinal direction will be considered prior to a path starting off into the general direction of the destination if the latter is longer than the first, even though a human would directly recognize the error in this approach.

The last algorithm presented in this lecture seeks to implement a mechanism that guides the pathfinding procedure toward the destination more directly, by exploring the nodes of the graph not only based upon the cost up to that point, but also upon the potential to reach the destination from that point.

Algorithms

A* Approach



Algorithms – A*

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

Core Idea: Next Slide

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

Algorithms – A*

Description

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

- 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 code.
  - 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.
  - Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least-cost paths in front.

Algorithms – A*

Description

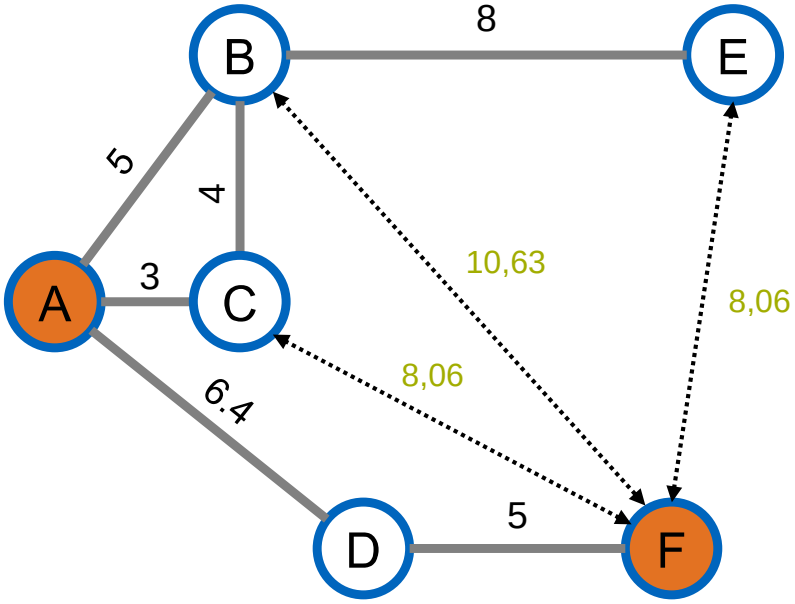
Algorithm – Core Idea:
From „Artificial Intelligence“ by Patrick H. Winston

- Until the first path in the queue terminates at the goal node or the queue is empty,
  - Remove the first path fr… …y extending the first path t…
  - Reject all…
  - 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.
  - Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least-cost paths in front.

Compare Lower Bound Estimate: Airline Distance

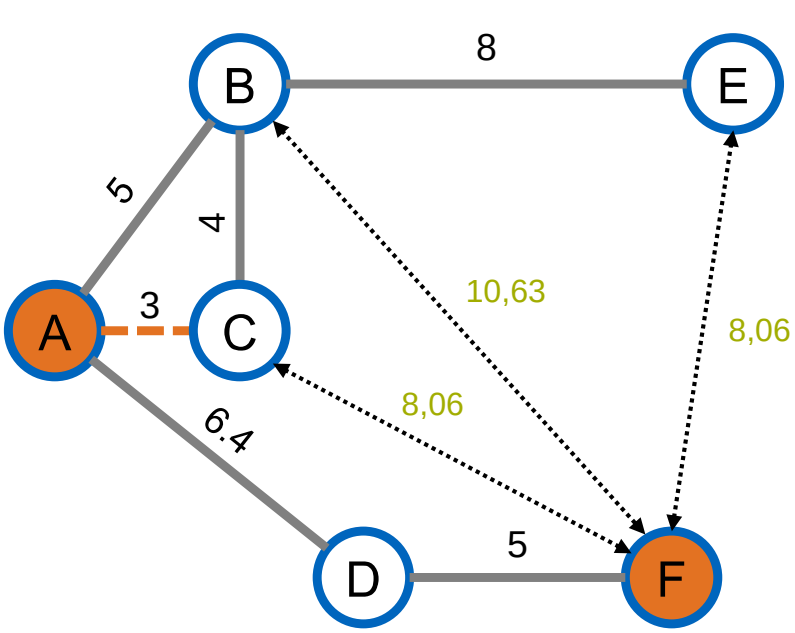
Algorithms – A*

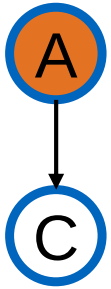
Pathfinding Example Graph



Algorithms – A*

Pathfinding Example Graph

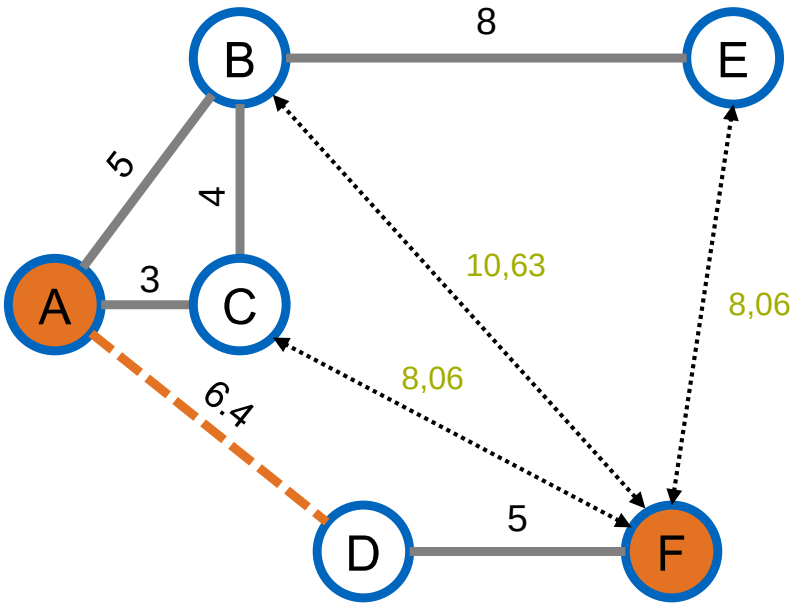


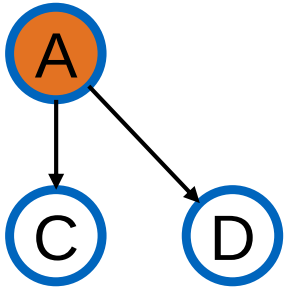


Airline Distance

Algorithms – A*

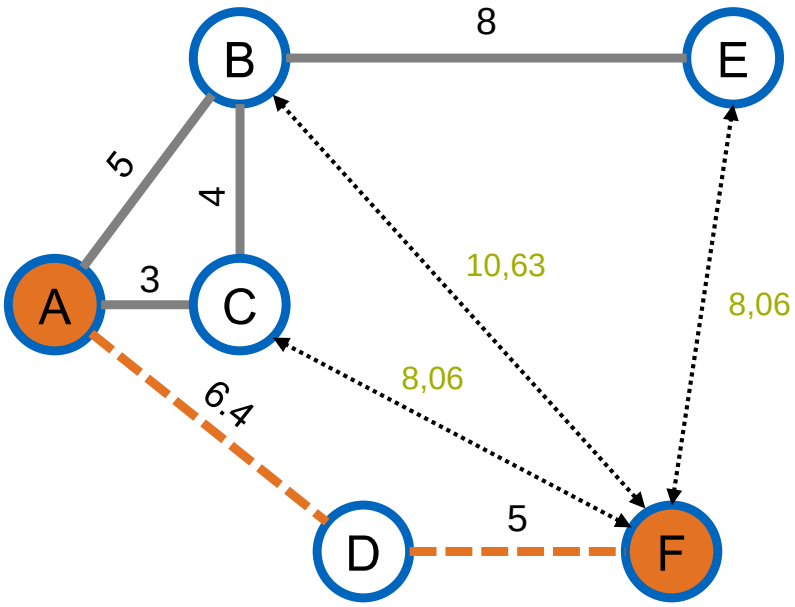
Pathfinding Example Graph

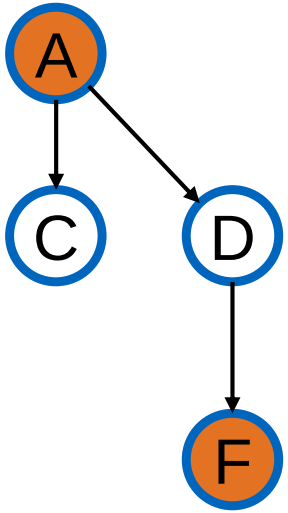




Algorithms – A*

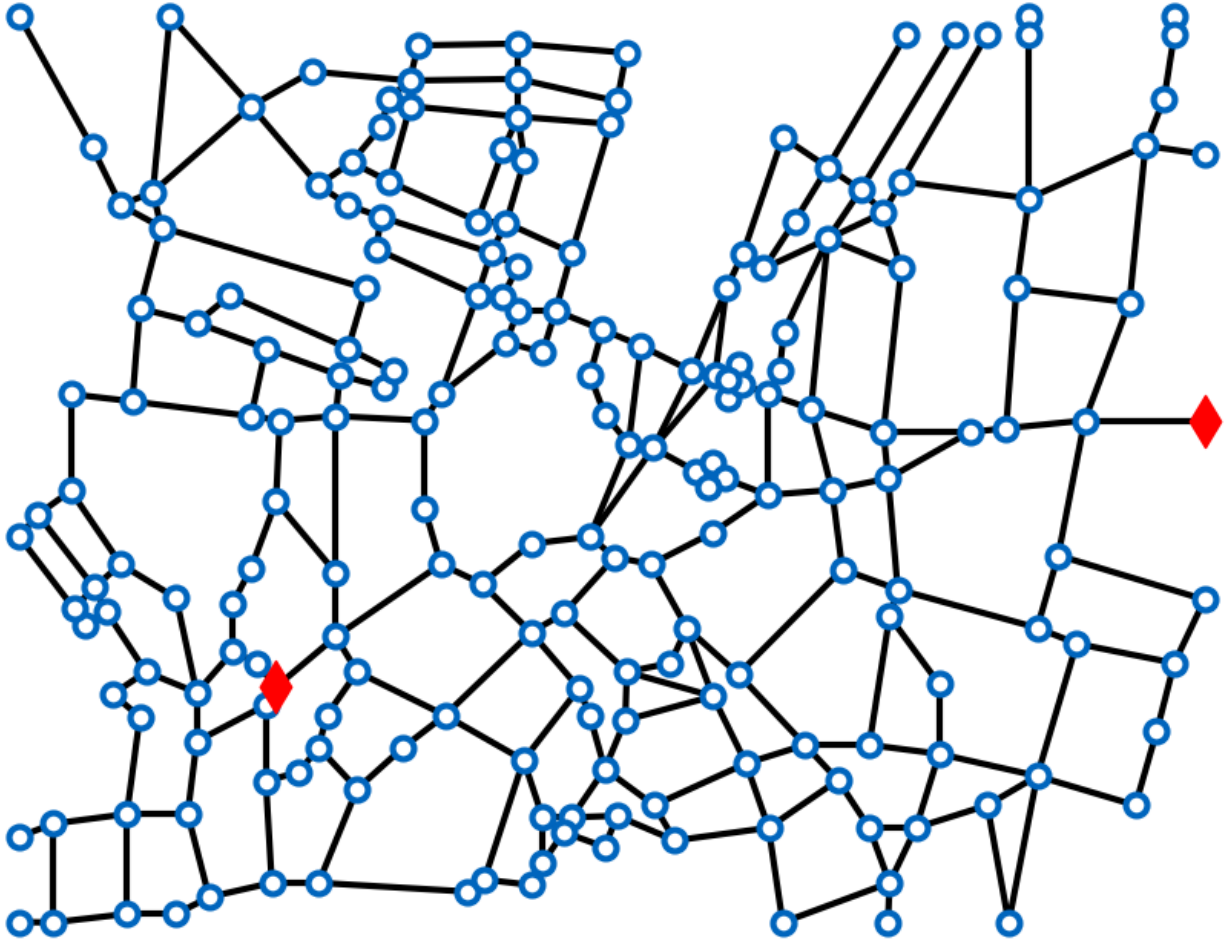
Pathfinding Example Graph





Algorithms – A*

Munich Demo



Additional Slides

The animation shows that A* converges much faster toward the destination than all of the previous algorithms. A* also favors paths in the general direction of the destination over paths that run in different directions. By doing so, the number of iterations is reduced, especially when working with large and highly connected networks in which the general direction is mostly the best one to follow (which holds for street networks).

A* explores all partial paths in the order of their potential to reach the destination with a minimum amount of steps. To guarantee an optimal solution, A* continues to explore the graph even if a feasible path has been found, because – unlike Dijkstra – it can not be sure that the first feasible path is an optimal path. In A*, the search process can only be terminated if there is no partial path left that could potentially reach the destination with less cost than the path already found.

Therefore, the estimate used in the algorithm has to be a lower bound estimate to not rule out partial paths that can potentially be shorter than a path already found. The airline distance is such a lower bound, because no set of street segments can be shorter than the airline distance between its start and the destination.

In real-life applications, A* or Dijkstra algorithms are used in navigation systems.

Algorithms
Attributes

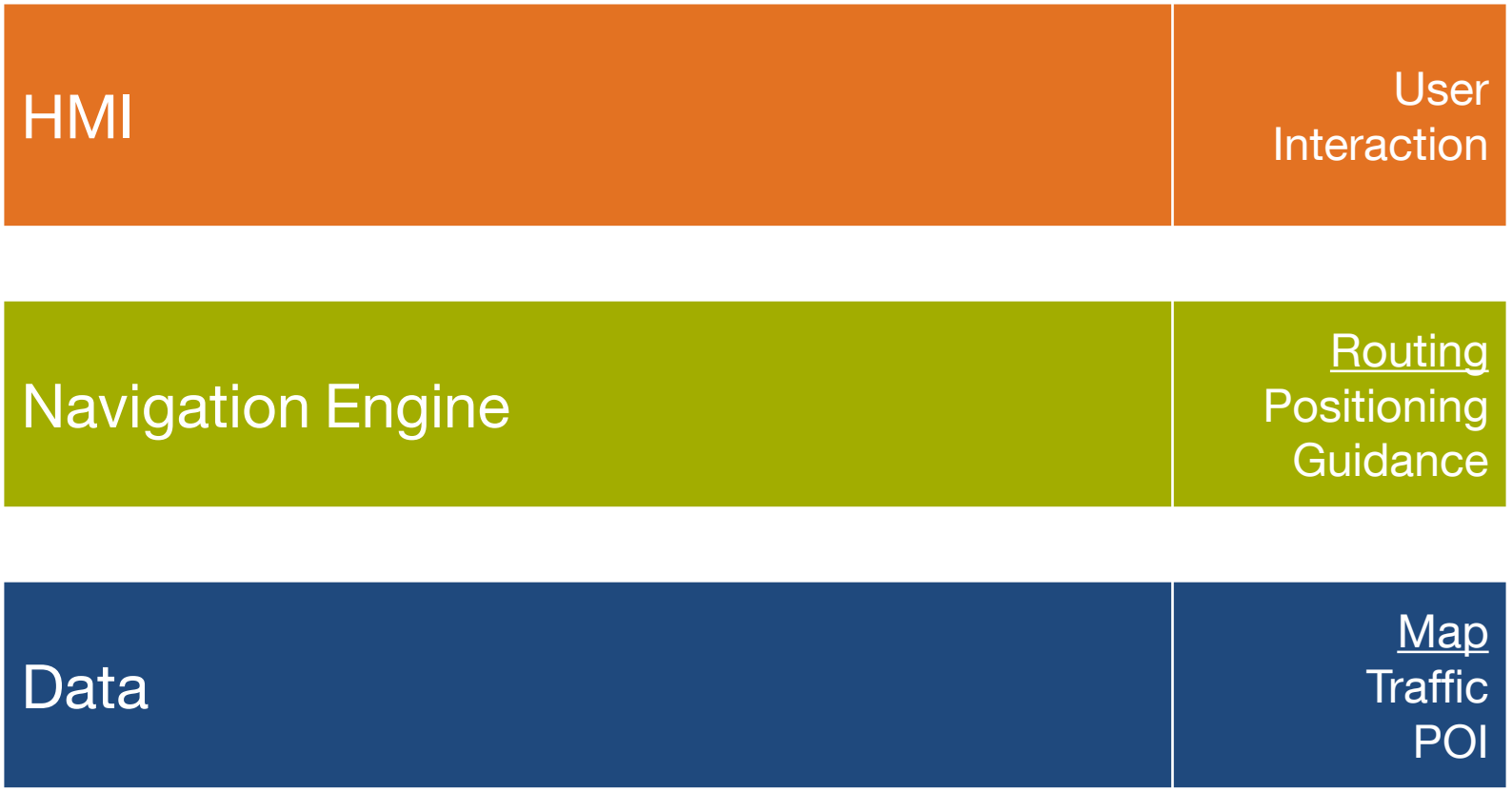
Informed
Best First, Dijkstra, A*

Complete
Breadth First, Depth First, Best First, Dijkstra, A*

Optimal
Dijkstra, A*

Application

Real life conditions



Additional Slides

The core part of this lecture dealt with the theoretic concepts of digital maps and pathfinding. These are generally applicable for real-life applications and with the knowledge from this lecture you will be able to work with productive navigation systems. However, when it comes to a large scale implementation of pathfinding outside of lab-environments, you will face a number of challenges that have not been dealt with in this lecture. To give you an overview over the most prominent challenges one has to overcome when bringing a pathfinding algorithm into productive application, the following slides make you aware of things that have to be taken into consideration on the Data and Navigation Engine level of a navigation system.

Application
Real life conditions

Number of Nodes:
~ 4 billion worldwide

Number of Ways:
~ 400 million worldwide

Vast Amounts of Data

Application
Real life conditions

Static Weights:
distance
cost
scenaric value
CO_2 emissions

Different Types of Static Weights

Application

Real life conditions

Data	Map Traffic POI

Dynamic Weights:
congestion
construction
ferry schedules
tolls
mountain passes

Dynamic Weights

Application
Real life conditions

Navigation Engine	Routing Positioning Guidance

Performance:
small RAM
dynamic rerouting
short query time

Performance

Application
Real life conditions

Navigation Engine	Routing Positioning Guidance

Positioning:
noise
signal loss
initial positioning

Inaccurate Positioning

Summary
What we learned today

Highway geometry, spatial information and traffic rules can be represented by a digital data model

---

Mathematical graphs can be used to model physical street geometry and information relevant for routing

---

Different routing algorithms are used to calculate a path through given mathematical graphs

---

Real life applications bring additional challenges not covered by the theoretical approaches covered in this lecture

---

Vocabulary and ideas