CS4386 AI Game Programming

Lecture 08
More on Pathfinding

Semester B, 2020-2021
Department of Computer Science
City University of Hong Kong

Improvement on A*

- With a good heuristic, A* is a very efficient algorithm
  - Even simple implementations can plan across many tens of thousands of nodes in a frame
  - Even better performance can be achieved using additional optimizations
- Many game environments are huge and contain hundreds of thousands or even millions of locations
  - Massively multi-player online games (MMOGs) may be hundreds of times larger still
  - While it is possible to run an A* algorithm on an environment of this size, it will be extremely slow and take a huge amount of memory
  - If a character is trying to move between cities in an MMOG, then a route that tells it how to avoid a small boulder in the road five miles away is overkill
  - This problem can be better solved using hierarchical pathfinding

Hierarchical Pathfinding (1)

- Hierarchical pathfinding plans a route in much the same way as a person would
- We plan an overview route first and then refine it as needed
  - The high-level overview route might be "To get to the rear parking lot, we'll go down the stairs, out of the front lobby, and around the side of the building"
  - For a longer route, the high-level plan would be even more abstract: "To get to the London office, we'll go to the airport, catch a flight, and get a cab from the airport"
- Each stage of the path will consist of another route plan
  - To get to the airport, for example, we need to know the route
  - The first stage of this route might be to get to the car
  - This, in turn, might require a plan to get to the rear parking lot, which in turn will require a plan to maneuver around the desks and get out of the office

Hierarchical Pathfinding (2)

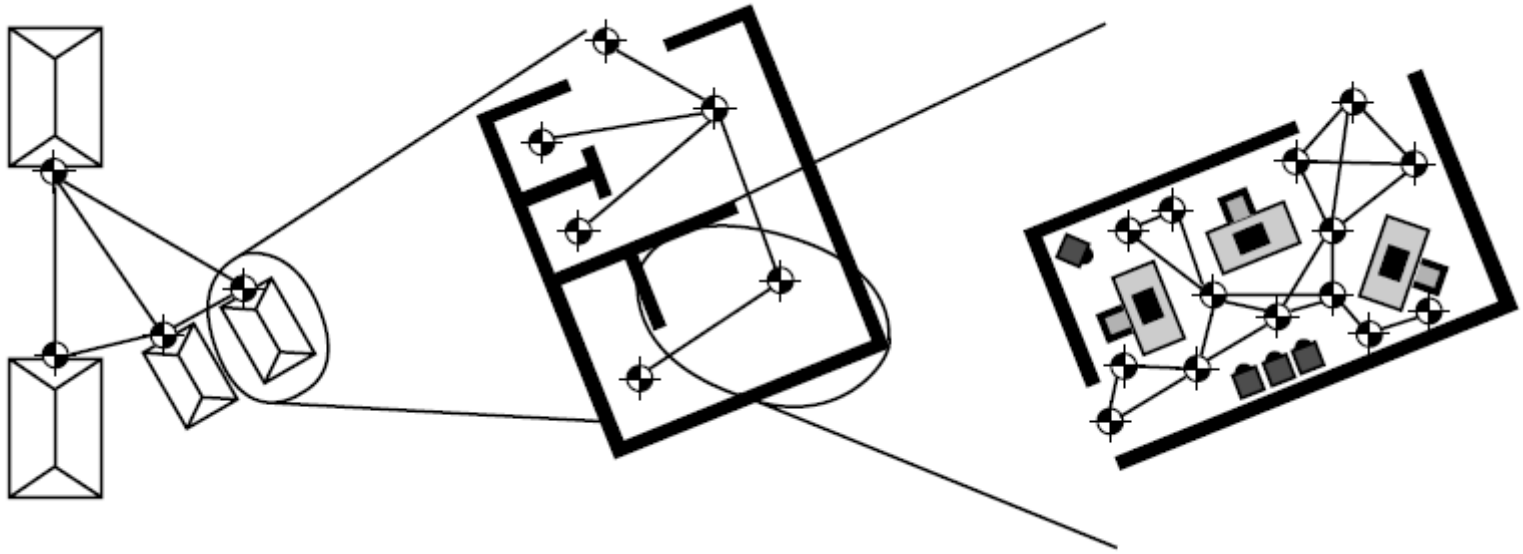
- To start with, we plan the abstract route, take the first step of that plan, find a route to complete it, and so on down to the level where we can actually move
- After the initial multi-level planning, we only need to plan the next part of the route when we complete a previous section
  - When we arrive at the bottom of the stairs, on our way to the parking lot (and from there to the London office), we plan our route through the lobby
  - When we arrive at our car, we then have completed the "get to the car" stage of our more abstract plan, and we can plan the "drive to the airport" stage
- The plan at each level is typically simple, and we split the pathfinding problem over a long period of time, only doing the next bit when the current bit is complete

Nodes (1)

- The locations are grouped to form clusters
  - The individual locations for a whole room, for example, can be grouped together
  - There may be 50 navigation points in the room, but for higher level plans they can be treated as a single node in the pathfinder
  - This process can be repeated as many times as needed
- The nodes for all the rooms in one building can be combined into a single group, which can then be combined with all the buildings in a complex, and so on
  - The final product is a hierarchical graph
  - At each level of the hierarchy, the graph acts just like any other graph you might apply pathfinding

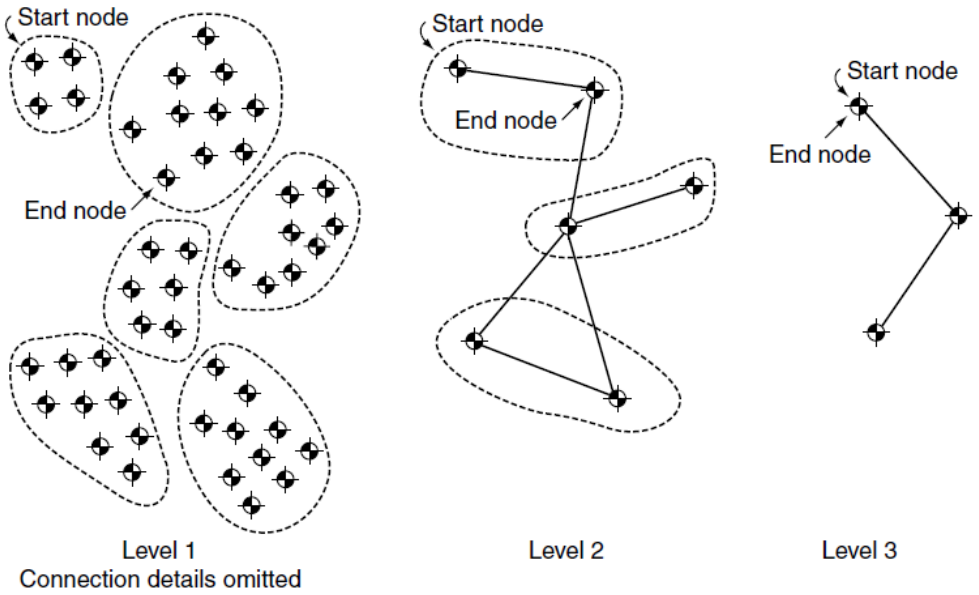
Nodes (2)

- To allow pathfinding on this graph, you need to be able to convert a node at the lowest level of the graph (which is derived from the character's position in the game level) to one at a higher level
  - A typical implementation will store a mapping from nodes at one level to groups at a higher level



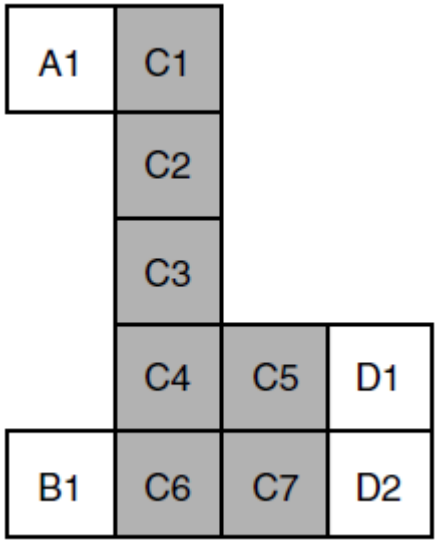
Connections

- Pathfinding graphs require connections as well as nodes
- The connections between higher level nodes need to reflect the ability to move between grouped areas
- If any low-level node in one group is connected to any low-level node in another group, then a character can move between the groups, and the two groups should have a connection



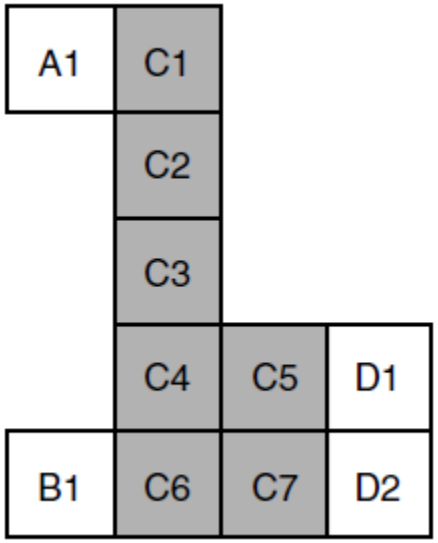
Connection Costs

- The cost of a connection between two groups should reflect the difficulty of traveling between them
- This can be specified manually, or it can be calculated from the cost of the low-level connections between those groups
- This is a complex calculation
  - E.g., the cost of moving from group C to group D depends on whether it is entered group C from group A (a cost of 6) or from group B (a cost of 3)



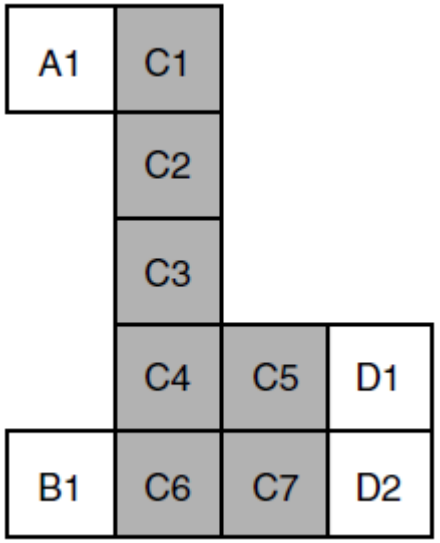
Heuristics in Calculating Connection Costs: Minimum Distance

- This heuristic says that the cost of moving between two groups is the cost of the cheapest link between any nodes in those groups
  - This makes sense because the pathfinder will try to find the shortest route between two locations
  - E.g., the cost of moving from C to D would be 1
  - Note that if you entered C from either A or B, it would take more than one move to get to D
  - The value of 1 is almost certainly too low, but this may be an important property depending on how accurate you want your final path to be



Heuristics in Calculating Connection Costs: Maximin Distance

- For each incoming link, the minimum distance to any suitable outgoing link is calculated
  - This calculation is usually done with a pathfinder
  - The largest of these values is then added to the cost of the outgoing link and used as the cost between groups
  - E.g., to calculate the cost of moving from C to D, two costs are calculated:
    1. the minimum cost from C1 to C5 (4)
    2. the minimum cost from C6 to C7 (1)
    - The largest of these (C1 to C5) is then added to the cost of moving from C5 to D1 (1)
    - This leaves a final cost from C to D of 5
    - To get from C to D from anywhere other than C1, this value will be too high
- Instead of taking the maximum value as in maximin, the values can be averaged as another heuristic

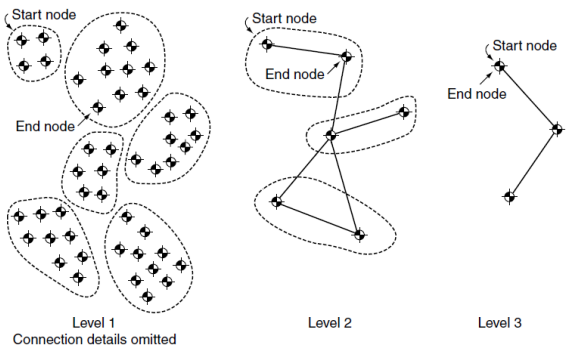


Summary of the Heuristics

- The minimum distance heuristic is very optimistic
  - It assumes that there will never be any cost to moving around the nodes within a group
- The maximin distance heuristic is pessimistic
  - It finds one of the largest possible costs and always uses that
- The average minimum distance heuristic is pragmatic
  - It gives the average cost over lots of different pathfinding requests

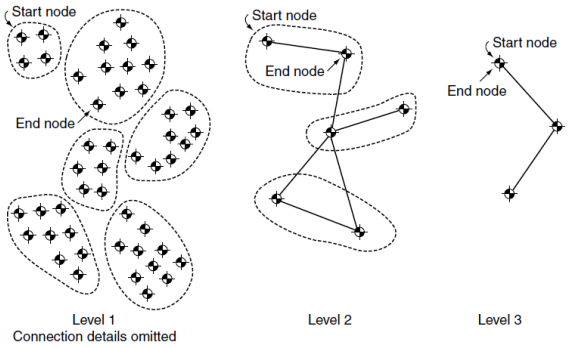
Pathfinding on Hierarchical Graph (1)

- Pathfinding on a hierarchical graph uses the normal A* algorithm
  - It applies the A* algorithm several times, starting at a high level of the hierarchy and working down
  - The results at high levels are used to limit the work it needs to do at lower levels
- Because a hierarchical graph may have many different levels, the first task is to find which level to begin on
  - We want as high a level as possible, so we do the minimum amount of work
  - However, we also do not want to be solving trivial problems either
  - The initial level should be the first in which the start and goal locations are not at the same node
  - Any lower and we would be doing unnecessary work; any higher and the solution would be trivial, since the goal and start nodes are identical



Pathfinding on Hierarchical Graph (2)

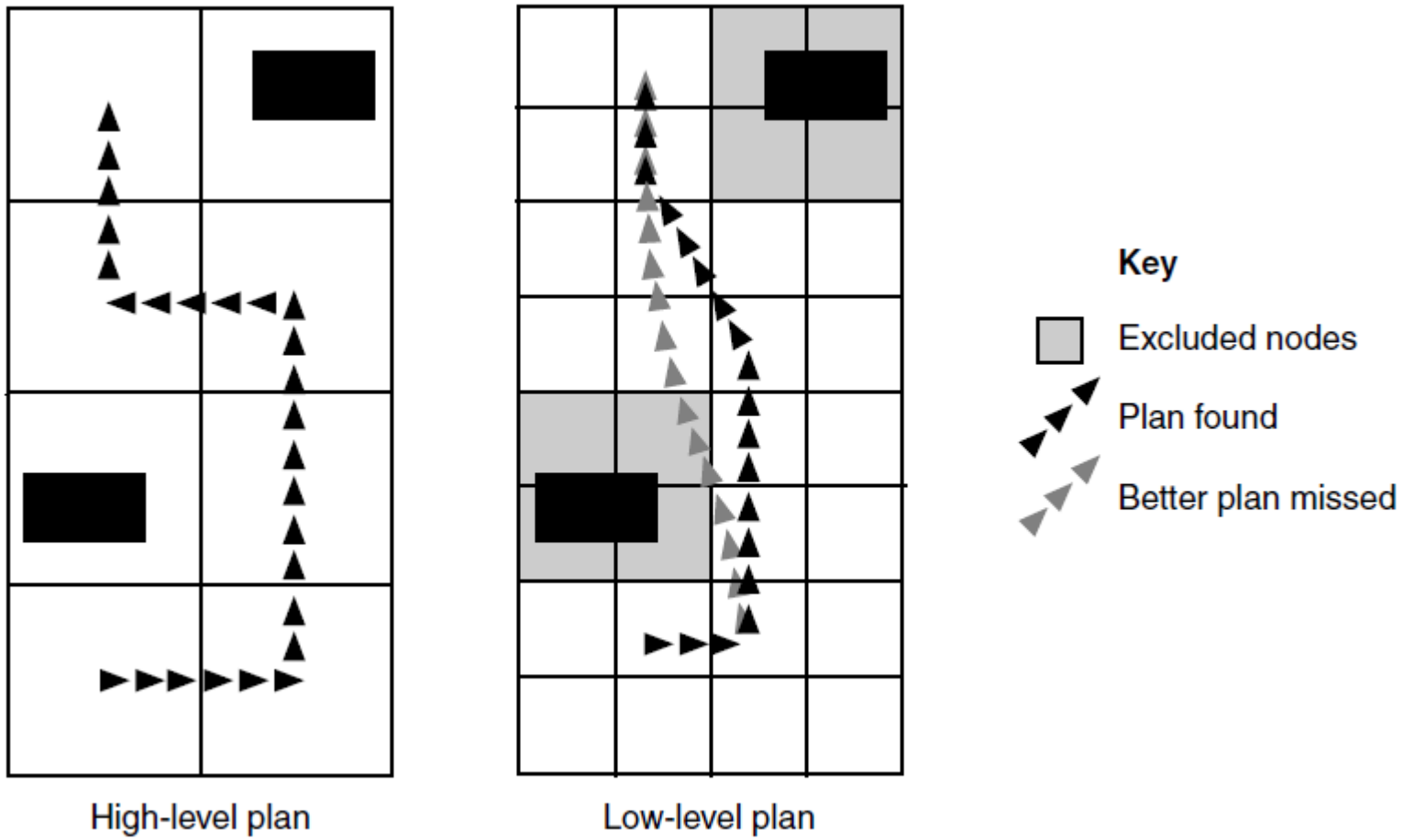
- Once a plan is found at the start level, then the initial stages of the plan need to be refined
  - We refine the initial stages because those are the most important for moving the character
  - Initially we do not need to know the fine detail of the end of the plan; we can work that out later
  - The first stage in the high-level plan is considered
  - This small section will be refined by planning at a slightly lower level in the hierarchy where the start point is the same, but the end point is set at the end of the first move in the high-level plan
  - This process of lowering the level and resetting the end location is repeated until we reach the lowest level of the graph



Hierarchical Pathfinding on Exclusions

- The previous approach plans for the path section by section
- In some applications, however, you might prefer to get the whole detailed plan up front
  - In this case hierarchical pathfinding can still be used to make the planning more efficient
  - The same algorithm is followed, but the start and end locations are never moved
  - Without further modification, this would lead to a massive waste of effort, as we are performing a complete plan at each level
  - To avoid this, at each lower level, the only nodes that the pathfinder can consider are those that are within a group node that is part of the higher level plan
  - It is not as efficient as the standard hierarchical pathfinding algorithm, but it can still be a very powerful technique

Hierarchical Pathfinding on Exclusions: Example

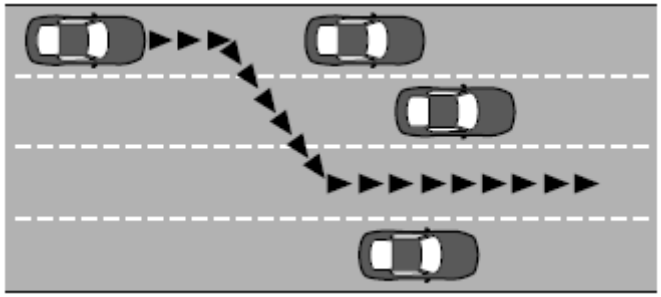


Continuous Time Pathfinding

- So far the algorithms introduced are for discrete pathfinding
- While this is powerful enough to cope with the pathfinding tasks required in most games, there still remains some scenarios in which regular pathfinding cannot be applied directly: situations where the pathfinding task is changing rapidly, but predictably
- We can view it as a graph that is changing from moment to moment
- One such scenario that requires more flexible planning is vehicle pathfinding

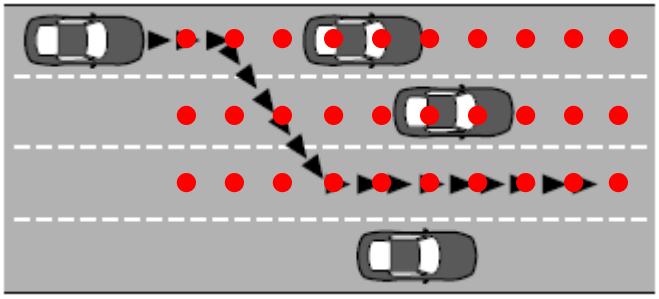
The Problem (1)

- Imagine we have an AI-controlled police vehicle moving along a busy city road
  - The car needs to travel as quickly as possible when pursuing a criminal or trying to reach a designated roadblock
  - We assume there is not enough room to drive between two lanes of traffic so we have to stay in one lane
  - Each lane of traffic has vehicles traveling along
  - We will not be concerned with how these vehicles are controlled at the moment, as long as they are moving fairly predictably (i.e., rarely changing lanes)
  - The pathfinding task for the police car is to decide when to change lanes
  - A path will consist of a period of time in a series of adjacent lanes



The Problem (2)

- We could split this task down by placing a node every few yards along the road
  - At each node, the connections join to the next node in the same lane or to nodes in the adjacent lane
  - Because the nodes are positioned in an arbitrary way, the player will see the police car sometimes make death-defying swerves through traffic (when the nodes line up just right), while at other times miss obvious opportunities to make progress (when the nodes don't correspond to the gaps in traffic)



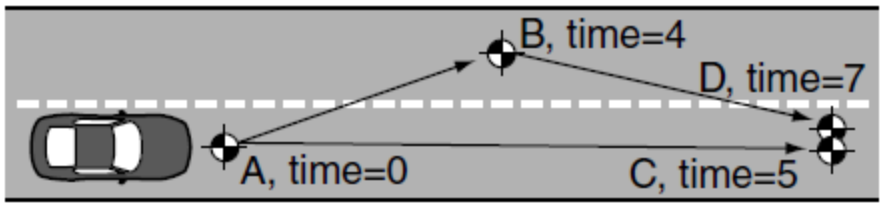
  - Shrinking the spacing of the nodes down will help but for a fast-moving vehicle, a very fine graph would be required, most of which would be impossible to navigate because of vehicles in the way
  - A* assumes that the cost of traveling between two nodes is irrespective of the path to get to the first node, which is not true in our situation
  - If the vehicle takes 10 seconds to reach a node, then there may be a gap in traffic, and the cost of the corresponding connection will be small
  - If the vehicle reaches the same node in 12 seconds, however, the gap may be closed, and the connection is no longer available (i.e., it has infinite cost)
  - The A* family of algorithms cannot work directly with this kind of graph

Algorithm

- The algorithm is in two parts
  1. We create a dynamic graph that contains information about position and timing for lane changes
  2. Then we use a regular pathfinding algorithm (i.e., A*) to arrive at a final route
- Previously, we mentioned that the A* family of algorithms is not capable of solving this problem
- To redeem their use, we first need to reinterpret the pathfinding graph so that it no longer represents only positions

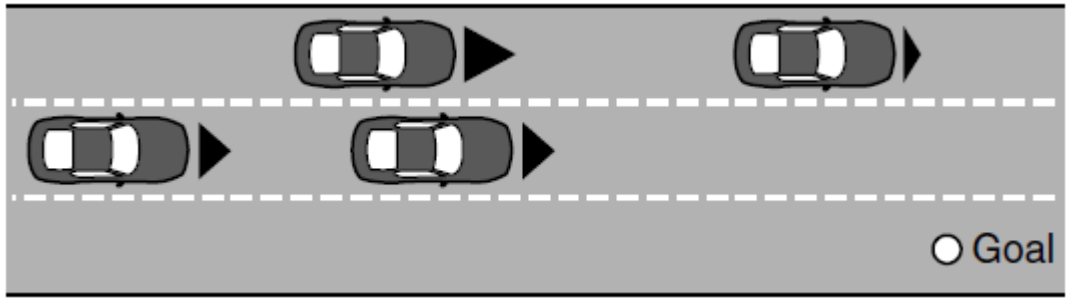
Nodes as States

- So far we have assumed that each node in the pathfinding graph represents a position in the game level and connections represent which locations can be reached from a node
- Rather than having nodes as locations, we can interpret nodes in the graph to be states of the road
- A node has two elements: a position (made up of a lane and a distance along the road section) and a time
- A connection exists between two nodes if the end node can be reached from the start node and if the time it takes to reach the node is correct
- Incorporating time into the pathfinding graph allows us to rescue A* as our pathfinding algorithm for this problem



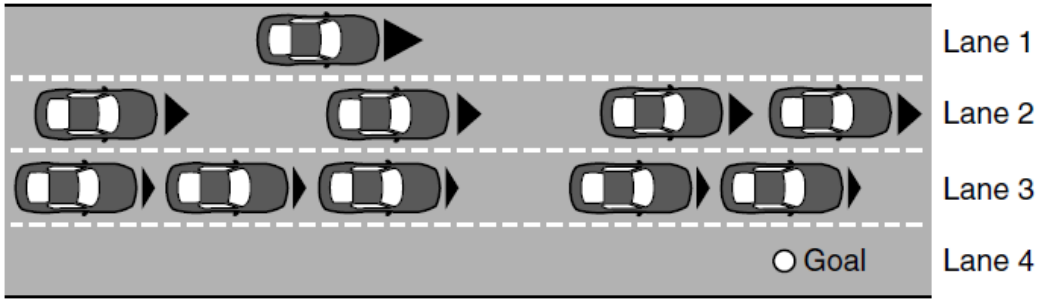
Size of Graph

- Now we have not only an infinite number of places where we can change lanes but also (because of acceleration and braking) an infinite number of nodes for every single place along the road
- We now truly have a huge pathfinding graph, far too large to use efficiently
- We get around this problem by dynamically generating only the subsection of the graph that is actually relevant to the task
- Example resulting options:
  - drive at full speed into the center lane as soon as possible and immediately out to the far side
  - brake, wait until the gap comes closer, and then pull into the gap
  - brake and wait for all the cars to go by



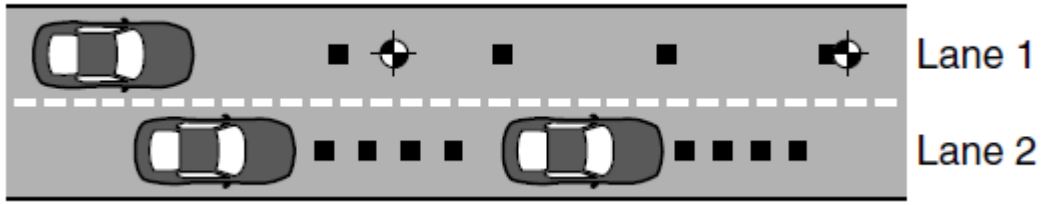
Assumptions

- We constrain the problem by using a heuristic and make two assumptions:
  1. if the car is to change lanes, it will do so as soon as possible
  2. it will move in its current lane to its next lane change as quickly as possible
- The first assumption is sound
  - There are no situations in which changing lanes earlier rather than later will give the car less flexibility
  - Changing lanes at the last possible moment will often mean that opportunities are missed
- The second assumption helps make sure the car is moving at top speed as much as possible
  - Unlike the first assumption, this may not be the best strategy
- In practice, drivers in a rush to get somewhere are quite likely to go as fast as they possibly can
  - Although it is not optimal, using this assumption produces AI drivers that behave plausibly: they do not look like they are obviously missing simple opportunities



How the Graph is Created

- Initially, the graph has only a single node in it: the current location of the AI police car, with the current time
- Outgoing connections from the current node are then added by examining the cars on the road



Because it is difficult to show time passing on a 2D graphic, the position of each car in 1-second intervals is marked as a black spot

Properties of the Graph

- The connections include a cost value
  - This is usually just a measure of time because we are trying to move as quickly as possible
  - It would also be possible to include additional factors in the cost
    - A police driver might factor in how close each maneuver comes to colliding with an innocent motorist
    - Particularly close swerves would then only be used if they saved a lot of time
- The nodes pointed to by each connection include both position information and time information
  - We could not hope to pre-create all nodes and connections, so they are built from scratch when the outgoing connections are requested from the graph
- On successive iterations of the pathfinding algorithm, the graph will be called again with a new start node
  - Since this node includes both a position and a time, we can predict where the cars on the road will be and repeat the process of generating connections

Weakness

- Continuous pathfinding is a fairly complex algorithm to implement, and it can be extremely difficult to debug the placement of dynamic nodes
- Even when working properly, the algorithm is not fast, even in comparison with other pathfinders
- It should probably be used for only small sections of planning

Reference

- Artificial Intelligence for Games
  - Chapter 4.5, 4.6, 4.8