in this lecture we consider ai problems where we are modeling the effects of choices we make with a longer term idea of what we're trying to achieve we will call it goal so unlike the say machine learning classification problem where we have an input and we we need to make a choice but that choice is simply right or wrong or maybe we'll assign probabilities but we want to assign the high probability to the right answer there's a sort of right answer that's defined in the training in these problems while we do have the problem of making a choice there's a couple absolutely critical differences uh most centrally we don't have a label as to which of these choices is right we have a goal and the goal is not achieved by one of these choices the goal is potentially achieved downstream so the goal is not an immediate thing so they have this the goal is not an immediate thing so unlike classification or regression we don't have a label for what this should be or if we work we can come up with a design that gives us a label but that extra work is is the topic here how do we convert this problem into a classification problem is something you could ask it's not on the face of it a classification problem because the specification of what we're trying to achieve is not a label on these out of these choices okay well we do imagine we might have the input which is this and this is like the current state of the world or the problem and we're going to take that and we want to choose between what well the the problem has to give us the choices as well and you can think of those as labels that we want to assign um but we're not going to be able to assign them by learning from a training data set because we have no such thing and um what we have instead is a goal downstream goal so there's a couple other critical features here one of which is ever present in ai and that is that this thing we're labeling the thing we're going to try to choose one of the options from is typically highly structured a complex object like a relational database a description of this current state of some problem even if the problem say you can't imagine the problem of solving a rubik's cube this is the description of the current state of the cube where every little qb is um there's an astronomical number of possible states even just of the rubik's cube and that's a pretty restricted ai problem so we're going to have complex structured state and a space of states that's absolutely enormous so that's what's going to differentiate this from a graph theoretic problem in algorithms where we want to find the shortest path from here to a goal and that's we actually have a graph theoretic problem but this element is makes it very different in character and lastly in today's discussion we're going to be assuming i put question marks here because we're going to relax this downstream but traditional ai first studied the problem of making choices where you know the results of the choices and they're deterministic so if we turn the top of the rubik's cube clockwise we know the new state that we're going to get that's not typically true in many say robotics problems or agent in the world problems many ai problems that are important to us we actually have a lot of noise and uncertainty in interactions another thing not listed here but that we're going to be assuming is full observability that we know the actual current state it's not like the back of the rubik's cube is hidden and we're not sure what what's on there that would be an interesting problem in its own right rubik's cube with partial visibility but that's not the kind of problem we're talking about here we have full observability so i've added that note so in this setting the effect of our action is to get a new problem of a similar form without any notion yet of whether we did the right thing or not so it kind of looks like this or we get a new problem looks the same we have to decide what to do or a new problem here or a new problem here and we we don't know whether we did the right thing here until we know what's right to do in at least one of these um and this is an ever-growing and exponentially growing and large structure that somewhere downstream there's a goal we're trying to reach the goal could be one of these states or it could be a predicate you apply to the state and says yes we like this one and so there could be a lot of different states that satisfy the goal we're trying to reach one of them we could also have costs on these edges some of these action choices we call these action choices we call this a successor state i'll put a note to that here so that's some of the terminology typically the idea is we're going to explore some part of this graph that's developing here in order to find the best or at least a path to a goal and that's the kind of method we'll be talking about in this part of the course so this leads us to a problem formulation and sort of a classic ai problem formulation where we're going to have some notion of possible states these are possible states of we might say the world or some problem and one particular possible state will be the current state or the initial state of our problem solving and from each of the states in the whole in the whole possible universe of states there are available action choices and the model must specify there must be a function that you can call pass it to any one of these states and it will tell you what the available action choices are and you must be able to use those in the model to figure out what the next state will be if you take that action so this might be a1 a2 and a3 three actions that are available if it's a rubik's cube these might be the six sides of the cube and each one can be turned in perhaps in either direction you might have 12 available actions also associated with the actions are costs they don't always have to be uniform there could be difference costs c1 c2 and c3 for the different actions of note here as i've mentioned is that we're currently imagining that our actions deterministically produce next states and these will be states of the same form as the state we were in but you know some some differences between them but the same structure so expanding that out imagine that we repeated those uh evaluations at this state and so forth we get a universe of reachable states that we might reach somewhere within that universe are some states we would like to reach and so our challenge is to find a sequence of actions that we can apply to this state the state we're currently in to reach any one of the gold states rather have a list of gold states we have a predicate that we can apply to states that says yes or no it is a goal state we may have some information about what qualifies as a goal state that we can use to try to make our choices to reachable state that's the goal is for me to have coffee i might use that information with knowledge about how the actions work to get to such a state so what you can notice is that this is a graph if we were to build if we were to explore all the possible states we could build a graph where the nodes of the graph are these boxes and some of the nodes are goal states one of them is our current state and we have costs on the edges and we want a path from the current state to a goal perhaps we want the lowest cost path and we could apply our graph algorithms so there are graph algorithms we will do a little brief coverage of that you would learn in an algorithms class and you'll be learning somewhat here that can find uh the a path or the shortest path within a graph but and those algorithms are efficient polynomial time in the size of the graph our problem is that our graph in ai problems is never a reasonable size virtually never so the graph itself is exponential or worse in size relative to the normal notion of problem description i can describe a rubik's cube to you but the number of possible states of a rubik's cube is astronomical and so the graph of all of its states with the edges being single terms of the sides is way way too large that you put the whole thing in memory and even worse for say chess chess is a problem with an opponent so that doesn't quite fit into this paradigm although you can begin to see that we could put it into this paradigm if we have two kinds of edges the edges where we choose and the edges where someone else chooses all right so the ai feature here is that this graph is going to be way too large to fit in memory so our standard algorithms algorithms are no longer considered tractable they're tractable and relative to the size of the graph but not relative to the size of the problem description which is so compact that it describes an astronomical graph in a short description i give you the rules of jess and now you have all the positions of chess in a graph okay so that's the nature of um ai search and it's relation to graph algorithms so it will help some of our methods if we keep in mind that we can track some data as we encounter these nodes or states in our graph two things that we'll refer to are first of all when we encounter a state as we're exploring we're going to have a path that led to that state because we're only going to explore from the initial state that we're interested in the current state or the source and as we explore from there as we encounter states we're going to have a cost to reach them so say we have a state s that we're exploring and we find there'll be some path from the start start state to the state s that we arrived there on and there will be a total cost of that path and we will call that g g of s function g maps s to the best cost we can achieve from the start state or if we're thinking of it as a data structure in the in a program it's the best cost we've seen from the start state so far that is the sum of the costs along whatever path led there among all the paths we've seen to s so far the best of those sums if we explore in the right order we will ensure that the best path we've seen so far is in fact the best path from the source to s and then g of s will actually be the cost along the best path and likewise when we're configuring out g of s for for some arbitrary s we'll have the path that led there at that cost and that path will have an immediate predecessor and parent of s will be that predecessor so there'll be some predecessor that led there along the best path and the parent of s will be this red pointer so by following the red pointers back we can get back to the start state along the best path we've seen so far so that's some data structuring that we will do and also mathematically a term will use this g of s as a mathematical function on the state that gives the cost to reach it from the start state along the best path so our fundamental problem is that this graph doesn't fit in our memory or even remotely so this is the fundamental problem we have to deal with in ai now this graph may fit in memory for many practical problems but those are typically not ai problems the graphical footage memory but even relatively toy ai problems like the rubik's cube the graph of all the possible rubik's cube states does not fit in memory and we don't have time to look at all the states so we have to solve this problem when we have some general grab bag of techniques that help with this problem there's no final answer to how we would solve this problem but the general obvious thing is we're going to explore the graph from the source like we're at some current state of the rubik's cube or some current state of the world and we can think about where we can reach in small numbers of steps and we can keep only that part of the graph that we have explored we don't have to put the whole thing in and some of the key algorithms that we're going to talk about that use this are breadth first search and uniform cost search we're also going to enrich those using the second idea into various forms of heuristic search called best first search and a star those techniques use in addition to what we've talked about they use an estimate of what's promising so we'll talk about um talk about that in more detail but the basic idea is that by looking at the definition of the goal and the properties in the structure of the current state we might be able to get an estimate of whether it's a promising state or not if we can quantify that estimate we can use it in our search as we will see a final idea that can be quite important is a space-time trade-off that even keeping this part we've explored in memory can be too expensive and what we can do is just go ahead and forget it and repeat the exploration with more and more aggressive parameters so we're going deeper and deeper into the graph but we're not remembering everything we did we're redoing it and this is a technique called iterative deepening and it plays off of the efficiency of a general graph algorithm called depth first search so iterative deepening is using depth first search uh to explore from the root deeper and deeper and deeper where the cutoff of the depth is used from these ideas up here so we're only exploring part of the graph according to these ideas up here but the part we're exploring is being explored depth first so we don't need all the memory to remember once we've defined these ideas we can look at how depth first search can be used to implement them so we're ready to start sketching a general purpose framework of an algorithm that will be able to accommodate all of these ideas so for starters for our general method we're going to remember every state that we've considered every state that's come into our consideration will remember them unless we're doing the iterative deepening um and as we um encounter them we'll keep track of the g value that is the cost of the path to get there and we'll also keep track of the predecessor remember what state led to the state so if this is our source and we've seen two different paths to this state here we'll remember the predecessor that led to the better cost and you can figure out what that is because each of these states will have g values and we can add the costs along these edges and see what the g value would be here on each path choose the better one and point the parent pointer back the way that's better now there's a subtlety in here and it's an important subtlety when you're implementing it's it's very difficult to keep all these subtleties and and caveats and alternatives in mind all at once so there's just a huge grab bag of search algorithms that make these choices in different ways and they often are ambiguously named so when we say best first search it's not always clear which best first search we're talking about what's the subtlety i'm referring to here well i could not fail to notice that i've reached this state two different paths i might actually have this state in the tree twice and not have detected it in that which case it'll have two different g values on it the one from this path and one from this path and two different parents and so this is a subtlety whether this grab bag of states that i'm keeping track of with various functions computable on them like g and parent whether that grab bag is a hash table so each state can be in there only once or whether it's just a table of states or you know states occurring in my memory lists and lists of states where the same structure can be in there twice and it's not clear which is better so uh obviously we'd like to know if we've seen this one twice on two different paths and just remember the better path but there's a cost to doing that which means we have to hash every state so we can detect that it's being encountered again and so there's this overhead cost of constantly checking if they've seen this state before against all the other states that we've seen which one is which approach is better depends on your application so we're not necessarily assuming that we detect this kind of thing this state could there basically could be duplicates in our tree and that that can be acceptable okay so either way though we're gonna remember all the states we're encountering possibly even with duplicates and the route that led to them and the costs along that route that's all of our general methods are going to do that now once we've done that we have this notion of expanding a state now this is a key term in state space search a state is expanded when we look at what we can do in one step from the state what action choices are there and where do they lead and so when we expand the state we add into consideration all the possible next states from that state so we're talking about in the first bullet that we have the certain data structuring around the states we're considering when we expand the state all of its next states go into consideration so we never need to expand that state again right because now all its next states are in consideration they're in our data structures we expanding it again won't do anything so the key process here is to keep expanding the region of states that are in consideration by expanding individual states once we've expanded them we don't worry about them anymore we're not going to expand them again so basically we're most interested in keeping track of the states we have not expanded yet we've not looked at what could happen if we took an action in that state it's the fringe of our search and so we take those states that have not yet been expanded called the fringe and keep them in some data structure i call it a prioritized queue but the priority need not be explicit it could be a first in first out queue in which case the priority is just age how long have you been sitting in our data structures under consideration but you haven't been expanded if you use a first and first out queue you'll be doing breadth first search more generally we can have a priority on the queue and if that priority within the queue is the g function you'll be doing uniform cost search which is another name for a very famous algorithm called dijkstra's algorithm so depending on what the priority is in the queue you get these two different very famous algorithms also if we use the priority in the queue that combines the g function and a notion of most promising so we say it's both a state that's easy to get to from the source low g value and it's promising with respect to being near the goal if we add those two notions together and use that as the priority we're going to get a star which is perhaps the most famous ai based search algorithm we will review those in writing all three of those breadth first search uniform crossing a star this is just foreshadowing okay so our general algorithm then within this paradigm is this star here this is the big skeleton for all of our algorithms big is being used as a term here for impact the impact on on this lecture is all in the side of this box is basically the sketch of all the different algorithms so we just repeatedly take the highest priority state that we have not yet expanded but has gotten into our consideration and expand it and when we expand it new stuff comes into consideration and joins our cue and then we do repeat that we just keep doing that until what until we get a goal of course now if we structure the notion of priority here properly we will be sure we have the shortest path to the goal at this point we're not always worried about that sometimes we're happy to find any path to the goal and we might be a little looser about how these priorities are maintained for instance the priority could just be how close the state seems to the goal even if it's actually a long ways it has a high g value so it's a long ways from the source it seems to be close to a goal so we'll just keep expanding near the goal um hoping to find a goal even though the somewhere else in the fringe there might be something that's going to find a goal closer to the source um that's i think i'll diagram that um so we might have our initial source and we might have something two options and these are both things that are kind of close to the to the source they're low cost to get to from the source but this one seems hotter maybe it seems closer to the goal so we keep expanding it and maybe we keep finding more states that seem close to the goal notice we're not finding the goal eventually we might find a goal this may not be the shortest path to the goal if we just ignored the fact that we were further and further away from the source but kept expanding this because it seemed close to the goal that structure ignores the possibility that even though this seemed colder it might have led to a goal sooner say there's a goal we never found that because we never expanded in there because we were so distracted by how hot these states seemed now if we if we allow that we're we're maybe prioritizing reaching a goal even if it isn't the shortest whereas if we set up our priorities right we'll actually be exploring this graph in layers not necessarily in number of steps but in cost layers so that we find this goal first so there's different algorithms these are i'm i'm i've recognized that without writing down more details this is fuzzy for you right now but i'm trying to foreshadow at a fuzzy level that your structure of how you choose priority here is going to give determine whether you have a guarantee that you find the shortest path to the goal first or ever and sometimes we don't care to focus on that we just want to get to any path to the goal so given this general structure for a search we get two different breadth-first methods closely related based on how we choose to define priority in this repeated expansion so our choice of priority gives us these two methods the simplest thing is just to use the oldest unexpanded state as the one will expand next so as we encounter states uh in these expansion we stick them in a queue we use the first and first out queue so the oldest one will come out first and we repeatedly extract a state from the queue and expand it putting its successors into the queue the places we can get by taking actions so we do that over and over until we find a goal so this is going to explore our state space in these layers for it will first explore all of the states one step from the initial state and then all the states two steps from the initial state and the order among these layers within the layer is not defined here because those would be ties in the queue what's a tie in the queue well we expand a node we get a bunch of next states successors choices that we had in taking actions from that node or state and the sketched algorithm doesn't say what order those go into the queue they they're added at the same time so it's unspecified and to be noted that there's no role in here for cost along these edges you can imagine they're all the same cost if there are costs we're ignoring them so we're not going to find the lowest cost solution it's going to find the lowest number of steps if all the costs are the same that will be a lowest cost solution so is this an effective method well the first and first out queue is very effective we're basically spending a constant amount of time on each of the states in the state space that we're going to encounter we're going to take the time to notice it and stick it in the queue we're going to take the time to extract it from the queue this takes a constant amount of time then we expand it get its successors constant amount of time stick them in the queue constant amount of time for each state and edge so you know if one state was to have a large number of successors you could imagine we have to do something per edge but this scales very nicely with the number of states and edges the problem is that in ai problems the space in which we're searching has an astronomical number of possible states we're not going to want to explore them all we're trying to focus only on where we can get from the initial state and get to the goal as soon as possible and not waste time on other parts of the state space but this size v just restricted to what we encounter in that process is going to blow up exponentially assuming we have a choice a branching factor b that gives the choice the number of choices right b is the number of choices that we see at each node so in this drawing it's it's one two three four at the start state and two at the others so assuming we have a branching factor that's not one then we're adding b times as many states into the queue at each layer as we did in the previous layer and after d layers we have b to the d states being put in and so here we're imagining we must go to d layers where d is the size of the solution the smallest number of steps to reach a goal so this is exploding exponentially and it's exploding exponentially in both time and space we're remembering this whole layer in the fifo queue so that whole layer is being remembered and then we'll expand all the nodes as we take them out and remember the next whole layer which is b times larger so then the size of the fifo q is growing exponentially and we're spending all that time to remove the states from it so it's both time and space exponential in the depth of the solution this can be viewed as an efficient algorithm from a graph theoretic perspective because it's linear in the size of the graph problem is an ai that size is exponential second breadth first method which suffers the same sort of problem but is going to use the costs is a generalization of this to where we have costs on the edges and so we're going to use as our priority the g function which i'll remind you is the total cost from the start state along the path that we've used to the state we're putting in the queue so if we're putting this state here in the queue this is the state s that's being put in the queue then g of s is the sum of these two costs if we discover s along a different path g of s will be the lesser of those two paths costs and so we're going to have in the queue this fringe of of states that we've considered but not expanded and take out the one with the lowest g of s value and expand that one now this is going to explore this graph in a different sort of layers where the layers have uniform cost now the drawing isn't perfect for this because in order for this layer to have uniform cost one of these edges would have to have zero cost which you can allow we definitely are not allowing negative costs here um that's not a cost at all but but so we could have a zero cost and these would both be in the same layer but the drawing is really supposed to indicate that that this is an area of low cost where we can get from the source in low cost whereas to go across one of these edges out of the layer would for this drawing to be accurate need to be one edge that's higher cost than these two edges so we will have explored this layer within this red larger red circle we will have explored this layer before we cross this edge and expand this if the idea is that these are low-cost edges and this is a higher cost these are higher cross edges so we're going to be exploring it in uniform cost layers this may actually be two layers if the costs here aren't zero because this has to be cheaper than this so we might have a layer in between here but the idea is to show the distortion here it's not any longer like circles around the source it's distorted in the direction of low cost and that's going to enable it to find the lowest cost solution first you can show it's pretty easy to show in analysis that the nodes selected for expansion here are increasing in g value over time so i've written that out for you the states as we've removed them one after the other from the priority queue that's managing the fringe will be removed in a non-decreasing g-value order we'll never suddenly discover a short path um so say we we've removed this vertex this is let's suppose this vertex is of all the ones in this fringe here around let's okay let me back up and set my context imagine that i've just completed this layer the red so i have unexpanded fringe of these states that are just across this layer this one this one this one on this one these other ones haven't even gotten into the fringe yet because i haven't expanded these i've just completed the two red layers then i have four vertices in my priority queue and one of them will have the lowest g value okay i'm going to expand that one i can make an argument i'm not going to do it here within the confines of this course but it's not hard to argue that i will never later expand some vertex that finds a shorter path to this vertex because of the way i've constructed this any vertex i expend later will if it discovers a path to this vertex we'll have a higher cost path you can sort of see that right if i'm expanding this one now at a certain g value any other path to here has to come through one of these other places and they have higher g values already so that path will be more expensive we're finding the shortest path first that's the point of uniform cost okay so states are removed in non-decreasing order there could be ties but it's not going to necessarily increase but it's not going to decrease okay this algorithm is actually also called dijkstra's algorithm in the algorithms community people out there sometimes make a big deal out of what are essentially trivial differences between these algorithms dijkstra's algorithm is usually viewed as processing the entire graph and finding the shortest path to every state in the graph but that corresponds to just what if we don't have a goal we just keep running this algorithm we'll find the shortest cost to every node in the in the graph in ai problems we don't imagine we can do that the graph is too large so we're going to stop when we reach a goal that's the basic difference and it's not really a difference at all so this is in my view this is dijkstra's algorithm the algorithm uses a priority queue priority queue is a queue where we don't take out the oldest node we take out the lowest priority queue and there are very efficient methods for priority queues that you can learn in algorithms classes so so it's not quite as efficient as a first and first out q it introduces a log term so it's not constant effort per node it's log of the number of nodes effort per node but still very efficient in terms of v and e v being the number of states we're going to explore and e being the number of actions that edges between states we're going to see as we explore so v is the number of dots in this and e is the number of lines or arrows and we're going to be very efficient in terms of vne not quite v plus e more like v log v e log v that sort of depends on how you implement the priority queue but very efficient but we still have the problem that this is an ai domain the number of vertices that it states in the graph is still on the order that we're going to actually encounter is on the order of b to the d in fact if the costs happen to all be one or all the same we are executing breadth first search it to generate to breadth first search degenerate is perhaps too pejorative it is breadth first search when the costs are all won then we can just ignore the costs and as i've said this use of bartique allows us to find the lowest cost path to a goal that will be the first with as soon as we see a goal in uh we don't even need to well we have to uh wait until we remove it from the priority queue but as soon as we expand a goal that's when we know we have the lowest cost path it does bear mentioning that in this algorithm because of the possible high variation between high cost edges and low cost edges the important moment when a goal is found is when that goal is expanded not when it's initially found and stuck into the queue because it might be later found again at a lower cost before it ever gets removed from the queue so just to show that in our diagram we can imagine this may be a very very expensive edge so this is sitting in our priority queue at a very high value along the way we're expanding we keep expanding this red circles that never gets to include this but it's including lots of other regions and one of them has a very cheap much cheaper pass to this vertex this vertex will get in the queue again along that cheaper path eventually and that's the one that will get removed and expanded so that if this is a goal we don't want to stop unless we're happy to find any path of the goal if we want to find the cheapest path of the goal we got to keep going until we remove it and expand it or we might find some other goal at a cheaper cost later all right so it finds the lowest cost path of the goal at expansion time when the goal is expanded and so i've written that out for you all right so that's it for our coverage of breadth first uninformed methods we'll later return to breadth first methods with information added in the form of like estimates of how far away the goal is we're also going to be talking a little bit about how we could avoid this time and space we can't avoid the time but how we can avoid the space burden of an exponential and make that space-time trade-off i referred to earlier in this lecture so we'll get more detail on that