in this lecture we're going to discuss optimization problems or problems where we're trying to minimize or zero out some kind of score conversely we could be trying to maximize it from us a search framework where we're searching through the possible ways of making the choices this is a widely studied problem where there are lots of different ideas and many of them provide performance improvements and in different domains different things tend to work so i'm going to review a variety of ideas these are accessible and easily understood ideas and you may have ideas of your own as well variants of these and combinations of these so many people have talked about these ideas and different uh slides that have been available and mousem at the time at the university of washington has collected many of these slides and produced a nice slide sequence that i i've selected slides from so side credit to all these people so this problem that we're going to talk we'll about local search local search we're going to review many different techniques as i just said and here's some of the examples that give an outline of the lecture hill climbing as and of course more generally gradient methods simulated annealing which is a stochastic technique and genetic algorithms which are inspired by by biology simulated annealing isn't actually inspired by physics a lot of the talk will be in the hill climbing area so what is this problem so this problem is that this particular side starts with what it is not it is not a study of a search problem where we're trying to find a path to get to a goal that is a problem where we make a sequence of choices paths the path to a goal in some space is a sequence of choices one made one after the other after the other but that's not the kind of problem we're studying here we're studying a problem where a single state in in the space that we're exploring has all the choices made it's a solution to the problem so for some kinds of uh problems that you might formulate as making us a sequence of choices um the solution makes all the choices and can be thought of as as all we need we don't need the path to it and here's the motivating example that we'll point at a few different times in this lecture which is called the end queens problem this is a diagram of the eight queens problem in this diagram we've put eight pieces on a chess board and the pieces happen to be queens and the goal is to put them on so that none of them are on the same row column or diagonal with each other happens to be how a queen moves in chess so none of them can attack each other but we don't need to know that for for our purposes just think about the rows and the columns and the diagonals so this queen doesn't share a diagonal with any other queen so this looks like a solution to me i have tried to check i don't see any row column or diagonal conflicts and this can be generalized to an n by n board there are exponentially many different queen placements as we increase n and exploring them all is prohibitively expensive so we're going to look at what's called local search techniques that explore something that's a proposed solution and try to fix it up to make it a solution by making small changes to it that's the general idea of local search what's local is we might take something that's not a solution and make a small change to it to try to make it a solution which basically amounts to what you might do if you put all the queens on the board and started moving them around to to remove conflicts now this is closely related this is looking for a goal right we're trying to satisfy a constraint find a way of putting the queens on that meet the goal but it's closely related optimization because we could say the score here is the number of conflicts in some sense and we're trying to drive it down we're trying to optimize it make the choices to get the lowest possible score we could imagine putting 12 queens on an 8x8 board and there will be no way to get no conflicts but we might want to still minimize the number of conflicts so these are closely related problems the techniques we're going to be looking at are local search problems where we only have to keep track of a single or small number of states of the of the problem in this case boards with queens on them we might just keep track of one and then we'll be changing it over time moving them around moving the queens around so we'll call that moving to neighboring states but we will not have to do something like a breadth-first search of the problem where we're keeping track of potentially millions of different configurations of that board now therefore we will mostly ignore what path we took to get like the queens may be in a certain configuration on the board and there there are some conflicts but we don't know how we got to that configuration we moved them around for a while we didn't keep track of how we we got to where we are and that means we'll use very little memory this is focusing on this on on fast analysis rather than on remembering a lot of things now this approach may seem relatively weak because it doesn't fill up memory with a lot of analysis of many different configurations but conversely it goes so fast through different configurations and and with the right design it can often find solutions to problems that we have no chance of systematically solving but note since we don't keep track of pads we have a real possible problem of traveling loops so problems suitable for local search will typically be optimization problems in the sense that every one of the states we're exploring will have some kind of score on how they're doing give you some idea of how close you appear to be to be solved and the goal is is always in these cases going to be able to to find the max or the min um we want to violate the constraints zero times or something like that we want to to optimize that score and often these kind of problems don't fit that well into searching through paths through the state space so here's some trivial approaches to such problems and these should not be ignored these are powerful ideas that end up taking part in our eventual solutions one is to just generate a state randomly throw those queens on the board if we design our notion of state right we can eliminate a lot of the placements of the queens on the board that are definitely going to fail for instance we can the notion of state can be a position in each column so we've already committed that there are no column conflicts because we're going to say the queen in the first columns at the top the queen and the second columns in the third row the queen and the third column is so forth so the state could be a sequence if it's eight by eight a sequence of eight numbers giving row placements for the eight columns we've already eliminated all the column conflicts we can generate one of those randomly and see if it solves the problem okay now we just repeatedly randomly generate if there's a lot of solutions to the problem we may have a good chance of eventually solving the problem now this is not a local search strategy it's not a search strategy at all it still has a role it can explore different very different parts of that search space very quickly we can also using the notion of neighbor neighboring solutions here a notion of neighbor might be a solution where all the queens are in the same place except one queen has been moved within its column that would be a neighbor and with that notion of neighbor we can randomly change the state pick a random queen and move it within its column that's a random walk and an interesting thing about both of these techniques is that if we run them long enough they will they will solve the problem if it's solvable this is what asymptotically complete means in the limit both of these solve the problem they do not solve it efficiently unless it's a very easy problem with many many solutions so then we add the idea that that we have score or some notion of quality on each state in the queen's problem we might count the pairs of queens that see each other along diagonals rows or columns and if that number is high that's a bad state we want to lower that number if we have that score we should only move a queen if it's improving the score this is hill climbing but in the case of the queens we want to drive down the number of violations so it's valley following or valley descending but we'll still call it hill climbing it's the famous term so here we're only we're going to make a non-random walk where we only take a step if we're improving our score so we're going to have a loop that continuously moves to neighbors improving the value so in the simplest form of hill climbing we're just greedily improving the value and we terminate when we can't improve it so here's a pseudo code for that and we can see that we have some kind of initial state to our problem and we we start with it and then we travel around this loop and each step we look at all the neighbors whatever we consider the neighbors are we we could be considering finding the leftmost queen that has any kind of conflict and moving it or find all the queens with any conflict and consider all the moves they can make [Music] or we could just consider any move any any single move of a queen within a column that some notion of neighbor will define the successors and we evaluate the score of all of them and take the best one and now we're just going to move current to be that one and loop unless the neighbor didn't improve so we're climbing we want to climb so if the value of the highest valued neighbor is not any better we just return the current okay so that's our pseudo code for basic hill climbing and to note it does not look ahead doesn't look where we might get after that you can look ahead you'll still be local search if you look ahead a little bit but this has been described by researchers as climbing mount everest in a thick fog with just go up and there are variants of course we'll be talking about variants in the rest of this discussion but generally an a common variant is to consider more than one of the successors not just the very best one and if there if there are ties in particular you have to do that and you'll you can randomly choose between those ties and we'll get into more of the ways of introducing randomness so this you know in this low dimensional diagram the state is running along one axis and the score is running along shown on the y-axis and successor relationship is being nearby along the x-axis we're walking and in this case we're trying to find a maximum so we're going to walk up now in these problems that we're discussing we have a discrete notion of state so there's not a gradient don't get confused with gradient ascent that there can be gradient and continuous techniques in this domain but we're talking about discrete techniques where we're just going to change something it's the score is going to change non-continuously and there's no gradient we're just going to look for a change to the state to a neighbor and here the idea is there's a neighbor to the left a neighbor to the right but we're going to go to the right because it's going up and we'll eventually reach here right and we will not be able to improve on this state by making a local change we'd like to make a non-local change over the year but we don't know that we're just stuck looking locally and so this algorithm will return this local maximum but notice if we were to randomly restart it we could easily arrive at a different place and a lot of the starting places will arrive here and so here's an example where we've we've taken four queens and put them on the 4x4 board and then we've made a local change to improve the number of pairs in conflict to here this queen is now not in conflict and then we made another local change and so on we still have a conflict though to fix this this won't need to be here and this one will need to be down here nope that's not right sorry this one will need to come down to here um there is one more local change to get to a solved state so this is making progress lowering the number of conflicts so it should be clear in this larger size we're not going to solve the problem by random walk we're instead going to need some notion of score that we're improving and what this diagram is showing you is if we if we count the number of pairs in conflict we're going to have uh what 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 i think um looking at rows columns and diagonals i counted something like 17 that's the claim on here as well pairs of queens that are either sharing a row column or diagonal when you have 4 along this diagonal that's a lot of pairs right but the numbers on the other squares are imagining if we took a queen and moved it within its column say we move this queen down here we'll remove four conflicts this pair this pair this pair and this pair will be fixed by moving this queen down and so all of this all of the squares on the board represent neighboring states here all of the squares that don't have queens on them because we could move the queen in that column to that square so this queen could move up to one of the 12s and that the 12s are the lowest numbers on the board so those are the neighbors in this setup the successor states that are of best value highest up the hill but in this case we're minimizing so lowest down the valley is to go to a 12. any one of these 12s could get a queen so there's many ties there so the formalization we're using here of the space is that we have all the eight queens already on the board and one in each column so any way of doing that is a state and we can move from a state to a successor state by moving any single queen to another square in the column and then we're going to have this notion of the score of the configuration or the state and i've been using a particular score that they're using in the examples here you need to define that score for your local search problem here it's the number of pairs of queens that are violating the constraint of same row same column or same diagonal and in this case we want to minimize so just as an example we have a a state it's not a solution and in fact as best i can tell looking at this example there's one pair that's violating there's one queen in every row and one queen in every column so it's just a question of diagonals and there's one violation so h is just one here so how does hill climbing do on this eight by eight problem well if we randomly generate the starting state and go to the best neighbor over and over it will get stuck on a local minimum eighty-six percent of the time but that's not as bad as it might sound because about one-seventh of the time it solves the problem and it takes on average four steps that's pretty darn good in that one seventh of the time because four these steps are very very fast so on one seventh of the time it solves the problem to generate a random state and climb up the hill and it solves it um you know in microseconds it takes even less on average to get stuck and be unable to improve so you can you know readily see that you could just randomly restart this and expect it to take seven restarts to solve the problem because this is 1 7 you know and finish much less than a second to find an 8 queen solution of course there's a non-zero probability of it taking 10 minutes it's just a microscopic probability um it's like the probability of being hit by a meteor but nonetheless it could because it's just random restart it could keep exploring the bad part of the state space random restart's very powerful that in this situation it's going to solve the problem for you it's going to get you into this 14 because each trial is fast so hill climbing this extraordinarily simple algorithm with random restart solves this problem very quickly even though there are 17 million states now 17 million states you know 8x8 we could have explored this with a systematic search as well so this is some visual thinking about the drawbacks we've already seen the drawback of a local maximum but plateaus are also of interest it can it can definitely be a problem even if the maximum is nice and unique and well defined that you may not be able to find it if your score often doesn't change you just end up wandering in a plateau not knowing where the maximum even is in fact for this technique you won't wander in the plateau because it doesn't allow you to make a move that fails to improve the score right the way we wrote that code it stops if if the best neighbor is equal so it can't even wander so that's something we can remedy we can allow sideways modes these would be called but notice that you have the possibility of getting in an infinite loop wandering and wandering here so we could say that if you can only equal the current score go ahead and take that sideways step and then you'll need to to do something to avoid an infinite loop like say we can take only so many sideways steps in a row if we do this with the eight queens problem we can now solve 94 of the problems from the initial random state without doing a restart but now it takes longer so you can decide whether you want to take this longer time to solve 94 of course then you can restart and you won't need to restart six times you'll expect to do it within two restarts so that's for eight queens i want to note though although we don't have the numbers here in these slides and i don't know the numbers the eight queens problem is solvable systematically 17 million states you can do that but if you were to take that to say the 16 queens problem now you're going to have too many states to systematically enumerate but it's entirely plausible that this technique will still be able to solve the 16 queens problem for you entirely possible i don't actually know um but that's the point of this is that we systematic search will completely break down and you'll still be able to do a stochastic local search taboo search is an idea of trying to break loops um and encourage exploration by keeping a taboo list so taboola's is just the states that you've seen most recently and you you can set as a hyper parameter of the technique how long the list is maybe you'll remember the the 1000 most recent queen placements that you've seen and then [Music] you simply it's like a first-hand first out cue add the new one the new state that you just saw and drop the oldest every time you explore and then you add this restriction that you can never go to a tabooed state so to be noted if this is just a list there could be a lot of comparison going on to check if your current state is in that taboo list you have to go down that whole list and for each item in the list you have to look at all the queen placements and see if they're this different and if it's a hash table you need maybe a good hash function for mapping for that's sensitive to these queen placements to to quickly find the placement you know in the taboo table if it's there anyways the design of this could really slow down the big advantage of local search so it's not clear whether this is going to save you or not if it's a short list it could prevent you from getting in loops on plateaus and as stated on the slide it improves performance in many problems so it's something worth exploring just i'm just saying you need to be careful how you implement it there's also an idea of getting something in between local search and systematic search and i find this idea quite attractive although there's some implementation i've heard where where you can think of this problem i'm at a current state and i want to find a nearby state that's better if i'm on a local optimum i'm going to have to flood the area if you think of it as a well if you get upside down what we're trying to minimize and my local optima is the bottom of the valley i want to flood that valley until i can get some water to run out over the nearby ridge and then i want to go that way and get out of the optimum so this involves performing a search around my current state until i find a better age function score and so that you know i'll be remembering all the states that i can reach you know i'll be doing something here's my current state and i'll be looking at all the things i can reach and i'll remember all of these then i'll try and continue to remember the fringe of this search you can look up breadth first search so then i basically have this fringe that i have to remember at all times and i keep expanding this fringe you know in order of how i could either expand it in order of how far it is the the state i'm expanding is from this my current this is my current state right in the in the center this is my current state or i could expand the most promising ones um it's unclear which is a better thing to do i'm trying to flood the basin and get out get to something that's better than the box have some score here i'm trying to improve on and i'm going to do a search bigger and bigger rings looking for an improvement as soon as i find an improvement i toss all this stuff and move the current state to that improvement so this will typically do prolonged periods of exhausted search but it's it's not doing only exhaustive search so it ends up being a trade-off between the exhaustive systematic search that you could set up that just looks at all the queen placements and the local search that doesn't remember anything so i i i find in for so common an appealing paradigm and it's certainly something you can try now our next topic is adding into this picture some randomness some more randomness as to what you might be doing we've already seen that you can just do random restarts and that's one of the things we can do we can also take a random step with some probability totally ignore the score sometimes and that might happen step after step so it's called a walk there's the length of the walk depends how many steps in a row you decided to take a random step you can take certain length of random step and that might be getting you out of a local optimum you can also just teleport out of it with a random restart and you can also have various forms of algorithm that when they find moves that improve the score they don't necessarily take the best one they could randomly select among the moves that improve the score and they can even set the probability distribution of that random selection so it prefers the best one but it doesn't necessarily take the best one and that's a stochastic algorithm that will sometimes do better than simply always taking the best one so we know that we can't just do straight hill climbing if they're a local minimum we may get stuck there we have to do something conversely totally randomly walking always works so we're trying to basically blend these two together with these ideas add more exploration into our generally promising algorithm that can get stuck so i've already discussed this hill climbing with random restarts and slide author recommends that if you want to do just one thing try this one and it's certainly something worth trying to see how it affects your performance and so there's some hyper parameters here some choices that you need to make how long you run before restarting you can run until you get stuck or you can run a fixed number of steps putting some of these ideas together we can also have a chance of taking a random step that's local it's not a random restart but it's a random step and a sequence of them could get you out of a local optimum so if we're just adding random steps then at each at each step we either take the greedy step or we take the random step and we'll have a hyper parameter p that's a basically our exploration probability well here one minus p is our exploration probability right it's like we go in the ice cream shop and with probability p we eat our favorite ice cream with probability one minus p we randomly choose a different ice cream because maybe we'll like one of them that's our exploration and of course you can do both these things and at each step you'll do one of the three things those very simple algorithms there are many many different combinations of these things you can try out so many of these ideas come together in a very well-known algorithm it's been around for decades that's inspired by physics called simulated annealing now the idea in simulated annealing is inspired by annealing crystals uh if you take a a solution that will crystallize as as the temperature goes down if you lower the temperature too quickly you'll get little pockets of nice crystals that collide with each other and the crystal and structure won't match when they collide like the crystallizing starts from a seed and from another seed and another seed and they start spreading out around the seeds and when the two spreadings collide at the boundary they can't make an ice crystal because they're not making a consistent pattern between them so you'll get these regions of nice crystal if you cool it too quickly if you cool it very slowly then what you'll find is it's still warm enough for one region to sort of spread into the other one and take it over and the two regions can end up merging so the idea in in the formation of a nice crystal is have an annealing schedule where the temperature goes down slowly at a high temperature the crystalline structure can change easily and as it gets to a lower temperature um we get if we do it slowly enough um we'll have let one region take over from another one and get one nice large large region of a nice large crystal now i'm no expert on making crystals so that's a pretty vague description so the idea is to have some notion of temperature now this slide doesn't really even mention temperature but it's going to get in let's see with this it's going to get in with this notion here that over time we're going to make it less likely to take a bad move so i know it has an intimidating name and this is also an intimidating slide with a long list of bullets the heart of this is right here so delta is the potential improvement that we're considering in in a move so notice we picked a random move we're not picking the best move we're just picking a random neighbor out of the neighbors and we're going to see delta is the improvement it could be negative if it's positive we'll just move so if there's a if the random neighbor we picked is better than the current state just go there but notice that may not be the best improvement available so already we've got some randomness there but otherwise we may even move there and whether we move there is going to be proportional to how bad the effect was if we got worse um we we will flip a coin on whether to go there and the bias on that coin will depend on how bad how negative delta is and it will also depend on the temperature which isn't mentioned on this slide but we'll get to it so that over time we'll be less and less likely to take these bad moves so the idea is with at a high temperature we'll be quite happy to take bad moves and wander around the terrain we'll have a bias to take good moves but we'll still be willing to take bad ones and as the temperature goes down over time we'll be rejecting bad moves we can also even make big moves when the temperature is high there's also a physical analogy to this crystallizing or annealing and that is imagine you've got a ball on a surface and you wanted to get it to the lowest point on the surface it will roll downhill but it's not going to get to the lowest point that way because it'll get stuck somewhere but if we shake the surface while it's rolling at the beginning we shake it pretty vigorously but then we shake it less and less over time the bigger shaking should help it get down to the lower regions and help it get out of local optima but now that it's in the lower regions we want to decrease the amount of shaking to hope that it can stabilize into the lowest of the low regions and over time we shake it less and less and get it to the lowest place okay and i've already talked about the crystallizing and the pseudocode may help i again want to say this should not be viewed as as intimidating as you know the name and the complexity of discussion is it's actually a very simple and easy to implement algorithm it does have the hyper parameter of this schedule you have to have some kind of schedule for the temperature it's kind of like the learning right schedule that we talked about in machine learning but or that you can talk about in machine learning but this so the temperature is going to go down over time and how it should go down you can research or try different hyper parameters for this schedule and when the temperature reaches zero we're done that means no more changes meanwhile though we randomly select a neighbor we don't have to try all of them we just need to randomly generate one and take that next neighbor and see if it's an improvement and if it is an improvement here we're climbing but if you're going down you would put less than zero then go to it let's see this is a very simple code and then the only subtlety really if it's not an improvement we have to compute a probability so this is where the temperature comes in capital t that's the current temperature and delta e is the amount of degradation so it's negative here so e to the some negative number that's delta e over t and you see as we get a very large temperature it's somewhat counterintuitive but as we get a very large since this is a negative exponent as this temperature gets very large this is getting very close to e to the zero which is one the probability that we take the step is going to one for large temperature okay so that's simulated and yelling and it's actually a very effective technique so well worth considering and just to review qualitatively high temperature we're more likely to take a locally bad move low temperature we reject those more and we decrease the temperature over time this has received quite a bit of attention and applied to many different things it was proposed back in the early 1980s as a technique for laying out vlsi and with the right temperature schedule researchers were able to prove that it will always find the global optimum in theory now that temperature schedule has to be very slow in decreasing in order to guarantee a global optimum that doesn't mean you can't have a very practical technique that decreases the temperature more rapidly if you decrease it very slowly then the algorithm can take a very long time to converge so that's that's a caution if you decrease the temperature too rapidly you may get stuck in a local optimum but of course you can then if you can finish that quickly enough you can use random restart so i think it's a promising technique with a more aggressive temperature schedule combined with random restart another promising option is what's called a beam search and the idea here is that when we have just one node in memory at a time the current node that's a bit of an extreme departure from the breadth first search that keeps all the nodes in memory we want something in between can keep track of some small set of nodes some k different states i'm calling them nodes k different ways of putting the queens on the board or something like that and initially they're just k different random starts and we look at all the successors of the k and if any of those successors is a goal of course we're done but then we take all those successors together and take the k best now notice that that's going to recruit like if if one of the randomly selected states was much better than the others it's going to recruit among the successors many of these k are going to come from one parent so it's not really that we're just doing k random starting points in parallel because one can recruit from the others in this step of selecting the k best successors so searches that find good states will recruit from the other searches to join them but we do have the problem that they can all end up on the same local hill and not actually helping and then we can help with that by not necessarily taking the k best but selecting the k successors from a probability distribution that's biased towards the good ones so we'll still have keep some of our diversity as many other ideas could be added here you could include a diversity measure in this selection and it starts to feel like we have some sort of natural selection going on when we get to here we're selecting from the beam to make the next step in the beam it's almost like a population involving another population with randomness biasing towards fitness and in fact this leads us right into the idea of genetic algorithms and this is our last idea so here we're not necessarily going to generate our successor states only from one previous state we can take two different states and combine them together much like sexual reproduction producing a child state so just to make this clear with our example we'll think of our queen problem again and we'll imagine now we're going to have a sequence of eight numbers representing the queen positions i've already described that before and we'll generate a population of random states to start with and our fitness function will be our score it needs to have higher values so we could do the number of non-attacking pairs if we want if we want it to be a true fitness function as opposed to an anti-fitness function and then we'll have a simulation of evolution and our evolution is going to have these three features um there's a random selection of who's going to get to contribute to the next generation and then there's crossover between that's the mating and then there's random mutations added this random selection is biased by the fitness that's where the fitness gets in so we see that in this example now just to make the string representation clear we're listing the row number of the eight queens by column row one row six row two row five one six two five seven four eight three okay so that's that's an e queen solution so here's the genetic algorithm in operation we take our initial population and we figure out the fitness so we apply some sort of score and this is the positive is good score 24 non-attacking pairs 23 non-attacking pairs 20 11 score for these and then we sample the the the ones that are going to we select by sampling from this population the ones that are going to contribute to the next generation we can do this many times over sampling two parents that are going to make two children here the way it's set up here they could just make one child and these two are selected at random from here but according to the these percentages that are defined by the fitness we normalize these to a probability distribution that's shown on the slide you should understand add them all up what fraction of the total is 24. that should be the the 31 percent so 31 of the time this one gets picked okay and then we pick a different one to mate so to speak and they the mating means that we line them up and pick a random point to draw and then read like this right here like this and that's the next generation l three two seven five two four one one well it's the original parent right here's the parent three two seven five two four one one so we've split the parent so that the the first three are contributing to the first child and the last five to the other this is not a direct copy of how recombination works say in humans and eukaryotes but it's inspired by that and so anyways we took the two parents and we and we made two children by taking the first part from one parent and the second part from the other parent we could have just kept one of these at random and then before we're done we also have mutations and this is there's a hyper parameter here what's the mutation rate um and it should be low because we don't want to totally destroy that whatever structure we had but with some probability we'll change something in here randomly and that gives us our next generation and we take that population put it over here and do it again so we have a lot of hyper parameters to control this we have to think about the beam width which is the size of the population so the one nice thing is that that these recombination steps the mutation steps are the local search but the recombination steps can take you to totally new parts of the search space pretty easily so it's kind of like blending in a notion of random restart that isn't random so it can be quite powerful it is quite a bit more um complex to implement and control and it has this negative of so many tunable parameters and so how well it works from one problem to another problem it's kind of an art to make it work so there's a very limited amount of literature of great results with these but it is a local search technique and and some people have do have a great time with these so you know it would be a lot of fun to implement and try i think so it's a bit mysterious that's our last technique there's a great many more out there and also these are very accessible so i think you can invent your own ideas quite readily and try them out