

Artificial Intelligence
Prof. P. Dasgupta
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture No- 16
Additional Topics

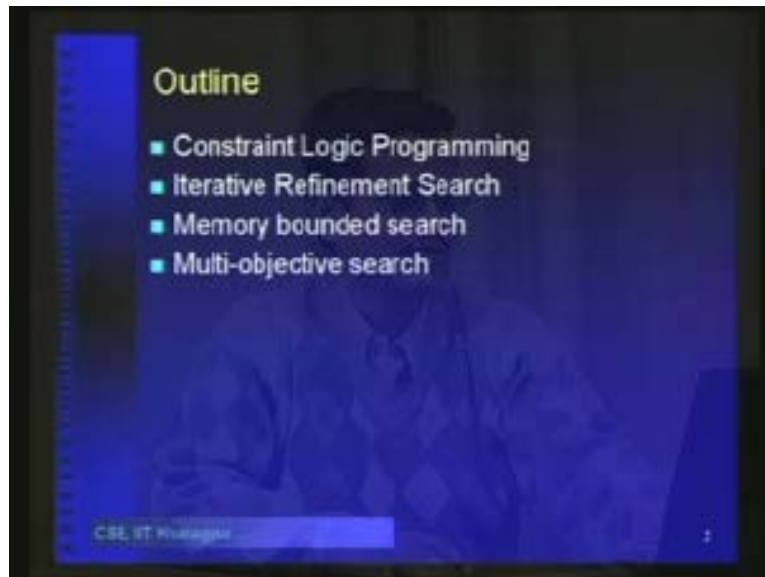
In this lecture, we will touch up on some additional topics which I had left out during the earlier lectures. We will cover up those things.

(Refer Slide Time: 01:14)



That should be what we will have for the mid-semester syllabus. Under additional topics, I have the following things to do. **In the** I will give you glimpse of what is called constraint logic programming. Then, I will talk about an area which is very important these days, called iterative refinement search, and just touch up on memory bounded search and multi-objective search. The last 2 topics are to a large extent, research work which was done in the institute here. I mean, memory bounded search was part of PPC's PhD thesis and multi-objective search was part of my PhD thesis.

(Refer Slide Time: 01:52)

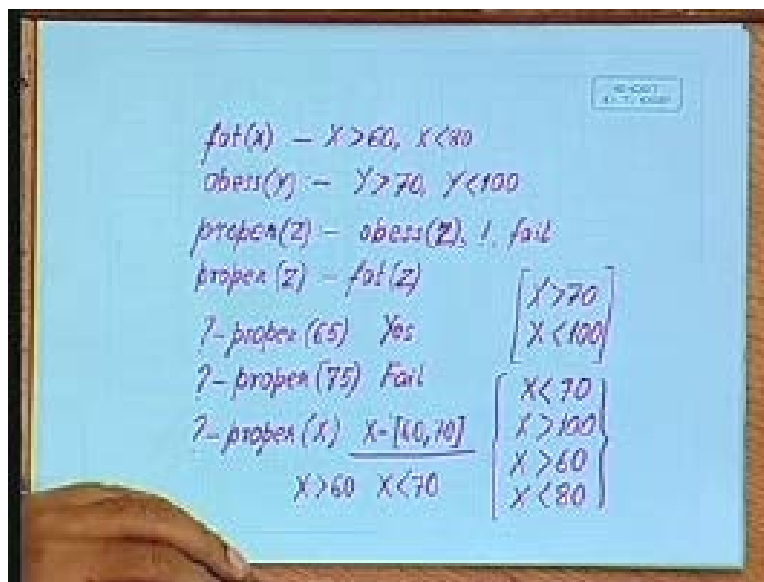


Let us start with constraint logic programming. Constraint logic programming addresses to introduce the notion of constraints into the logic programming framework. As you have seen, that prolog allows you to have certain constraints, like x greater than y , x less than y , etc. But when you have numerical values which are uninstantiated, then prolog tries to instantiate them by selectively starting with 1 number and keep on increasing that. It will try instantiating x for x equal to 1, x equal to 2, x equal to 3, like that. What constant logic programming **is going is** does is, instead of having an instantiation based approach, it maintains a constraint stack at the back end.

So, every operation is pushing certain constraints into the stack. Each instantiation, as we have in prolog, is a constraint which is an equality constraint. When we are trying to say that x is equal to y , we are instantiating x with y . It is a constraint x equal to y that we are implying by that. When we replace x by 4, then it is x equal to 4 that we are using. But when it is x greater than y , it has to be the constraint x greater than y that has to be pushed into the stack. So, in the back end, constraint logic programming is going to maintain such constraints in a stack. Whenever the constraint stack becomes inconsistent, that is a point where we backtrack and then try out other testing.

Finally, when we have finished the entire traversal and we have reached all solved nodes through some path, where the constraint stack did not become inconsistent, then that represents a solution. And the solution is going to give for every variable, the range within which the constraints are satisfied. It is going to give you ranges instead of actual values; it is going to tell you that, if for these ranges, the solution will hold. Let me take a small example to demonstrate constraint logic program. Suppose we say, that if a person's weight is x , if the weight of a person is x , then we say that that person is fat. If x is greater than sixty and x is less than eighty.

(Refer Slide Time: 12:09)



We say that the person is obese, if y is greater than seventy and y is less than say 100. Then we say, we have 2 rules which define what is proper. We want to say that a person is proper, if that person is not obese but fat. How do we write this, that the person is not obese but fat? This is a case of negation as failure. We will first try obese and then cut fail, so this is obese then fail- sorry, z . Now, if we ask that is 65 proper? How is this going to work? It will try this 1 first and check whether obese 65 is true. When we go

here, we find obese 65 is not true. Therefore, we backtrack and try this one, and we get fat 65, so we try this one, and yes, fat 65 is satisfied.

Therefore, it will say yes. It will say yes if we try 75. What it is going to do is, going to try this 1 first- obese 75. Yes, obese 75 is true, it **goes to cut** goes to fail. That means this whole thing, and along with this proper also, is discarded and it returns fail or no? Right. But all this did not require anything of constraint logic programming. Suppose we ask proper x; now what is going to happen? This is going to try out this 1 first, and then, we will have obese x, so that is going to instantiate x greater than 70 and y x less than hundred and we have this cut and fail. It will have to infer that it is the range outside this that is going to take us here.

When it is not this, so, when is not this, when x is less than 70 and x is greater than 100, then we will go here, and then, we will succeed with this fat x, provided x is greater than 60 and x is less than 80. Now, it is going to resolve these constraints now. Here, if we resolve these constraints, what are we going to have? (Student speaking). I wrote this, right? So, x is greater than 60 and x is less than 70. So, what this is going to return? Is x, should ri lie in the range of 60-70? This is the power of constraint logic programming takes place text.

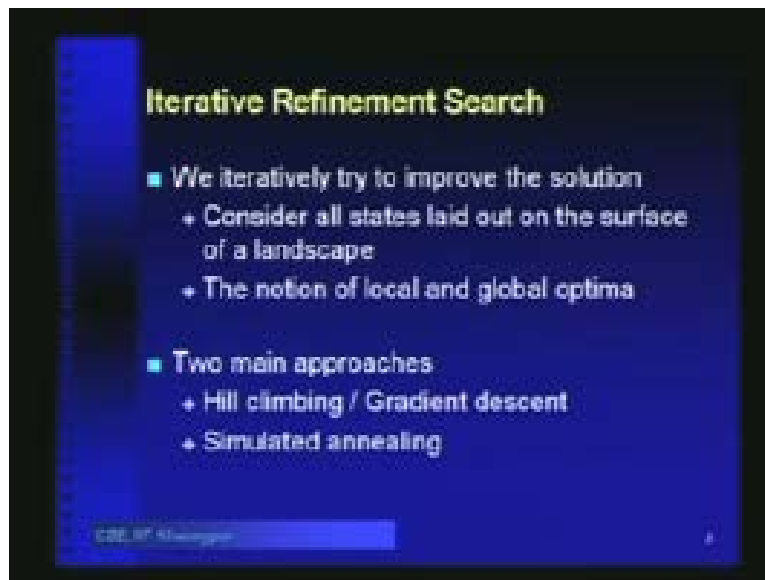
What it actually returns is that, it does all this. It maintains all this in the constraint stack, and then reduces the constraints to the variable ranges. Now, there can be multiple sets of ranges- combinations of ranges- for the solution, so it is going to find that out and return it. (Student speaking). Yes, if it cannot resolve the cut, then it has to actually take the negate of that ranges and then deduce that what is going to be the case for the other case. When am I going/not going into this cut? It will reduce data, then use that as part of the constraint stack to go into the next one. So, as you can see, that the reasoning that it has to do internally is non-trivial, right?

But there are today's solvers that accept constraint logic programming. And further, there also has been research on how to integrate optimization into the logic programming

framework. Can we represent optimization problems in logic programming? So, for What we have seen is, we have seen constraint satisfaction problems like 8 queens, and other problems like quick sort, etc., solved in logic programming. With this- with constraint logic programming- you can solve several other problems, but you still cannot solve problems like TSP, you still cannot solve game playing problems, because those will involve costs and we have to optimize cost.

There has been recent research on how to integrate optimization predicates- predicates which will involve an optimization criterion, and you have to solve the predicate in such a way that the cost is minimized. There has been some recent research, in fact, part of it done by our group also, on how to do this. Moving on to the second topic, we will look into iterative refinement search. We go back to the search procedures as we were studying previously, and have a look at iterative refinement search.

(Refer Slide Time: 19:20)



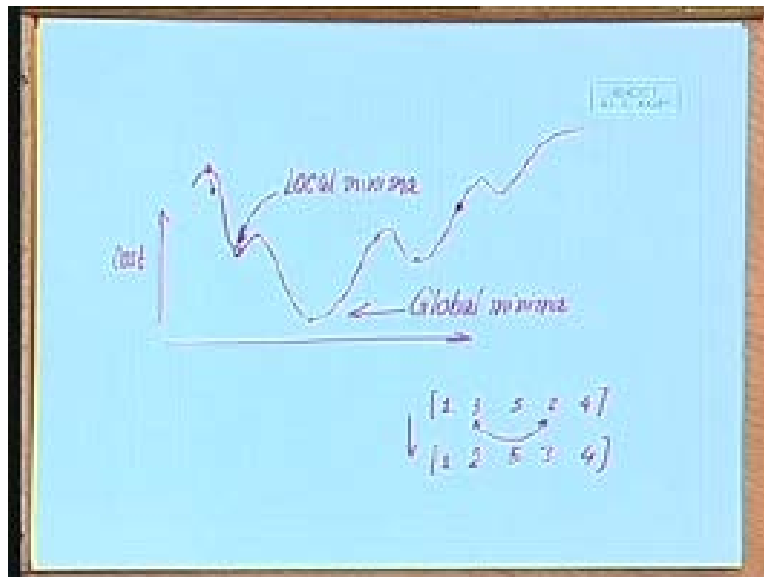
So, what is iterative refinement search? In iterative refinement search, what we will do is, we will we have an optimization problems and this is good for those optimization problems where you do not have where every point in your state space is a solution. How

can you have that kind of scenario? Take for example traveling sales person problem. Any permutation of the set of cities represents a 2. There is no other constraints, there is no other constraints. If you take a permutation of the set of cities, then that represents a 2, so iterative refinement search is going to try to tweak around your permutations, so that your cost improves. So, we have a set of solutions. We start with some solutions and try to improve them progressively, until we can improve no more. This is a basic philosophy of iterative refinement search.

These algorithms, like simulated annealing and genetic algorithms, they come under this category of iterative refinement search. Iterative refinement search can also come with constraints, which makes it more difficult. It could be the case, that not all permutations are solutions. In that case, you will have to also check that iteratively, we are also eliminating the constraints that are there. So, you have to keep satisfying the constraints and doing the iteration and the refinement, which is more difficult. We will first study how to do iterative refinement, where every point in our state space is a solution. So, the objective is to improve the quality of solutions through search.

Now, there is 1 way of looking at this thing, which is quite useful. What we think is that, we look at the solution space or the state space as a surface, as a landscape. And we are looking for, let us say, the minima. We want to minimize the cost, so this is the set of solutions and this is the cost, so this direction is the cost. Each of these points represents a solution, as I was saying, that we have just moving in the solution space, and for many problems, this is quite easy to do. Now, in TSP, what could be the kind of operators that we use? Take any pair of cities; exchange them. Your current tool could be 1, 3, 5, 2, 4, right? I take any pair of cities and just exchange them, so that will give me another tool- 1, 2, 5, 3, 4. This is an operator which changes you from 1 solution to another.

(Refer Slide Time: 18:45)



Then you check whether the cost has gone down, so, if we are initially at this point and suppose we say that we will apply the operators only when the cost goes down. If we do this, will we solve TSP? We may not, because you might end up in what is called a local minima, whereas, what we are looking for is the global minima. And you can get stuck in the local minima, because exchanging a single pair may not be able to improve your solution quality beyond this point. If you reach a point where exchanging any pair is not increasing the quality of your solution, then you could be stuck in a local minima. So, at that point of time, if you probably exchange more than 1 pair together, then you could have further brought the cost down.

We have to see- how do we get out of the local minima and ensure that we are in the global minima? The solution mechanism is very simple. We have some operators and those operators take us from 1 solution to another, and the result of that those operators may increase the cost, it may decrease the cost. But we have to use them judiciously, so that we are able to reach the global minima. How do we do that? **There are** Firstly, I will just look at 2 main approaches- these are 2 philosophies, actually. And then we will see that there are various categories of algorithms which use these philosophies. 1 is called

hill climbing or gradient descent- that is what I was just now mentioning, that if you keep on iteratively improving your solution, then that is called hill climbing if you are interested in the maxima, and gradient descent, if you are looking at the minima.

And let me explain what is the philosophy of simulated annealing. You remember what is annealing? You raise the temperature of the metal and then cool it down very slowly. I mean, compared to that, the other alternative is quenching, when you bring down the temperature suddenly. And if you recall, that if you do annealing, then the final structure to which the metal stabilizes is much more stronger than what you get by quenching. This tells us that if you are actually trying to minimize the energy of the final configuration, then annealing is probably a better method, where you reduce the temperature slowly. Simulated annealing derived from this basic philosophy and does exactly that.

What we are going to do is, we will define a thing called energy and the energy will be controlled by a parameter called temperature. This temperature is a parameter which is a random factor, which will tell us when to apply an operator which increases the cost. When the cost decreases, we will take it, but **when the cost** when I try a operator and find that the cost is increasing, will I take that operator or not? And when, in a higher temperature, I will take it more often- when in a lower temperature I will be more conservative, the idea is that I have operators which can improve the cost; I have operators which can make the cost worse.

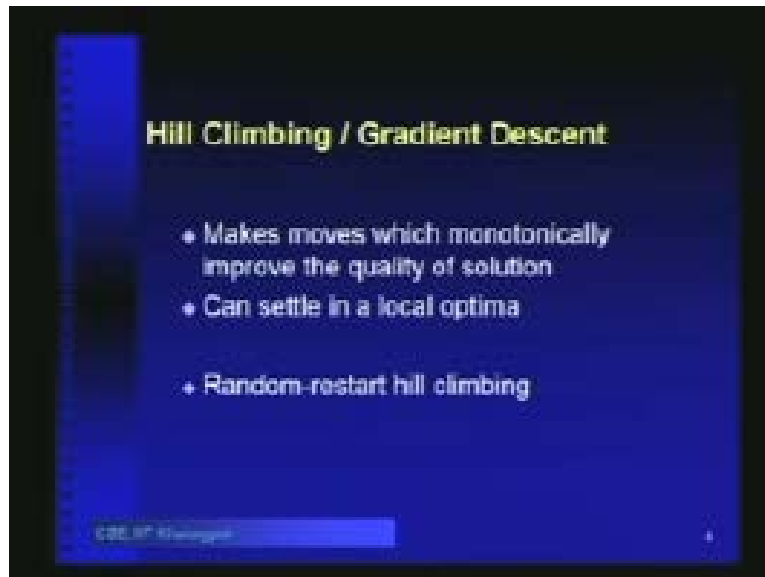
At a high temperature, I am going to use the ones which make the cost worse, also with a higher probability. And then slowly, I will bring the temperature down, and as I bring the temperature down, the operators which make the cost worse will be used with less and less probability. When I reach temperature 0, I am only doing gradient descent; I am not using those operators which increase cost any more. Now, there is a nice analogy if you look at this surface here. Suppose I give you a ball, and I ask you- that can you keep shaking this landscape, so that the ball settles down into the global minima? How can we do that? Now, think of the following algorithm- I will initially shake it really hard, so

then, the ball can jump from higher altitudes to lower altitudes, and it can also jump out of lower altitudes into higher altitudes, because the shaking is vigorous.

And then, slowly, I reduce the vibration, so there will be some point of time, where it will be able to jump out of the local into the global, but not from the global into the local. There will always be a certain vibration, which will be sufficient to jerk you out of this into this, but not enough to jerk you out of this into this. That is the point where you are moving into the global minima, and then, if we do this very gradually, then we are certain that it will go into the global minima, and also mathematically, we have been able to establish that if this reduction is very gradual, asymptotically spanning across infinite number of temperature cycles, then you will indeed converge on the global minima.

Now, what should be this function? We need a function which helps us in computing the probability of applying the operator when the cost increases. This, again, derives from the Boltzman equation, which actually describes the temperature's effect on the annealing procedure in the annealing of metals. We will use the same function; people have used the same function and with good results. Let us look at the algorithm now, okay? Hill climbing or gradient descent we have already seen, which monotonically improves the quality of the solution but can settle in local minima. There are variants like random restart hill climbing; if you get stuck in the local optima, just randomly start from another point, then do the same.

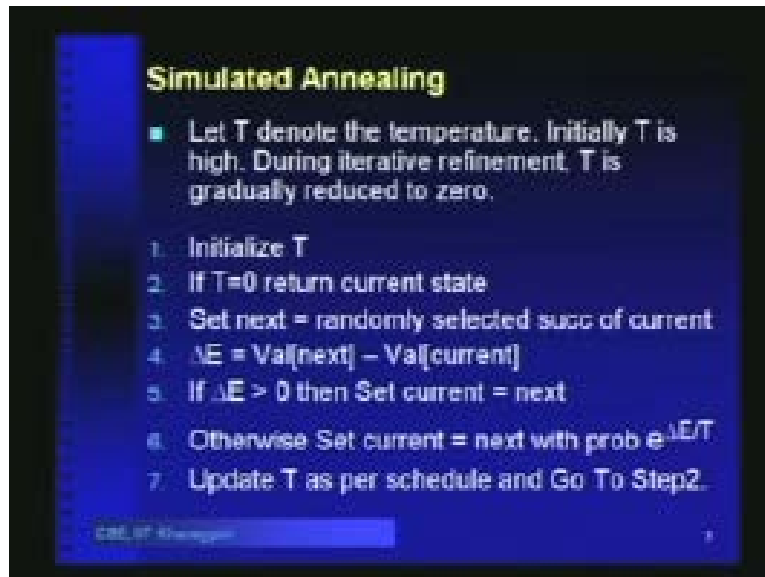
(Refer Slide Time: 25:04)



In simulated annealing, this is what we will do. We will take a parameter called the temperature. Initially, temperature is high. During iterative refinement, T is gradually reduced to 0. Initialize t . If T is equal to 0, return the current state- that is when we have done. Otherwise, select a randomly selected successor of the current state and compute the change in energy. What is a change in energy? It is the value of the next solution, where this operator would have taken us minus the value of the current solution. Now, if this is negative, that is where the cost has decreased. So, if it is negative, then the cost has decreased. Then we will take it.

If the cost has increased, which means that this delta is positive, then we have to do something about it. Now, depends on whether we are doing a minimization problem or a maximization problem. For the other one, it will just be the opposite. Here, we are doing a maximization, so if delta is E is greater than 0, then set current equal to next, which means that if the cost increases, then we just select that next, because we are trying to maximize it.

(Refer Slide Time: 29:00)



Simulated Annealing

- Let T denote the temperature. Initially T is high. During iterative refinement, T is gradually reduced to zero.

1. Initialize T
2. If $T=0$ return current state
3. Set next = randomly selected succ of current
4. $\Delta E = Va[\text{next}] - Va[\text{current}]$
5. If $\Delta E > 0$ then Set current = next
6. Otherwise Set current = next with prob $e^{-\Delta E/T}$
7. Update T as per schedule and Go To Step2.

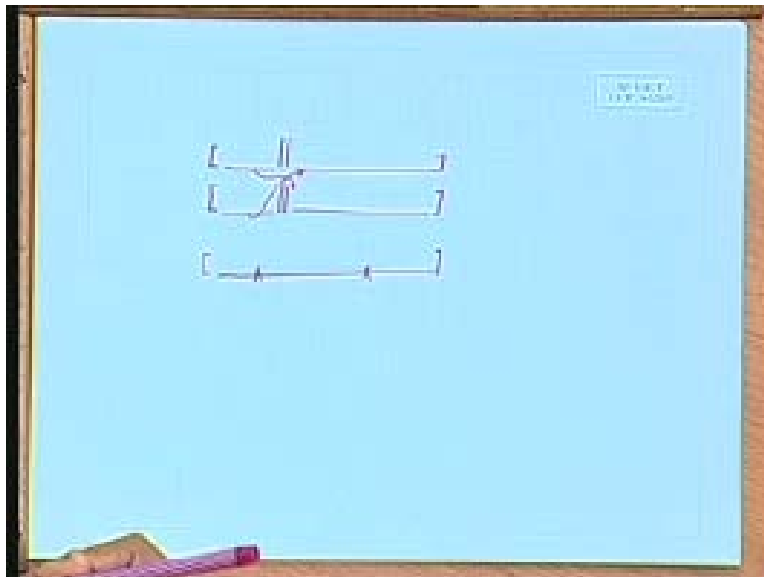
Otherwise, delta is negative, **otherwise delta is negative** so we set current equal to next, with probability of E to the power of ΔE by T , because that this delta is negative, so this value is actually between 0 and 1. Because this delta is negative, and we have biased it with the temperature. This is the Boltzman function. We have biased it with this T . When T is very high, then you will see that this probability is higher. If T is 0, then this probability is 0 or asymptotically approaches it. As limit T tends to 0, this is going to be 0. And then, we will update T as per the schedule and go to step 2.

So see, that the algorithm is very simple and it works really well for a whole lot of problems. You can try writing out something like the TSP. With this formulation, you will see that it really comes down to very good solutions quite rapidly. And you are guaranteed, however, of the optimal solution, only if your temperature schedule is asymptotically spanning across infinite steps, and with very little change in the temperature in every cycle. Now, **when you have** do any of you have an idea about what genetic algorithms are? You must have heard about genetic algorithms. Yes or no? What we do in genetic algorithms is, we have a set of- that is also another way of iterative refinement search.

We have a set of solutions; we have, at every point of time, a set of solutions. Each solution is a string- like, for example, our list of cities is a string. And then, you can do a set of the set of operators that you have, are namely of mainly of 2 types. 1 is called a crossover. A crossover would mean that if you have a string like this, 2 strings like this; crossover between these 2 strings would be that we cut this string at some place, this string at that same place, and then, this, I will join with this to get another chromosome, and I can take this with this, to get- so, this is a traditional crossover that we had studied in school when we studied genetics.

And then, there is also a mutation operator, and the mutation operator is one which allows you to simply take 1 string and mutate some parts of it- change some part of it. Now see, if we did only crossover, then there was a possibility of getting stuck in local minima, because we would become very much dependent on the initial set of solutions and you will not be able to get out of that initial set of solutions and go into different other types of solutions.

(Refer Slide Time: 30:42)



Mutation actually allows you to mutate the chromosome, so that you can also venture outside this. Now again, when you are doing with when you are working with genetic algorithms, what are we doing? We have some cost, which tells us the goodness of an individual chromosome as a solution. It also tells us the goodness of the whole set of solutions that we have at this point of time. Each step is performing some crossovers and mutations and checking whether the quality of solutions that we have, has improved or not. As you can see, that there are, again, the possibilities of getting stuck in local optima for the same reasons.

Now, if we have a reasonably large population of solutions on which we are doing these mutations, then, it has been shown even without doing the things like simulated annealing, if you keep on doing this crossovers and mutations, then you are likely to find a very good solution. Now, the complete stochastic analysis of genetic algorithms is a very difficult thing to do. So, people have mostly studied genetic algorithms by empirically dealing with different problems and modeling them as genetic algorithms and seeing how they perform.

And in many cases, they perform really well. That is why this has become reasonably important discipline of search. There are some initial results; like there is theorem called Schema Theorem- just take a note of it the Schema Theorem- which gives you some initial results about when genetic algorithms are likely to converge, but we are not going to go into the details of this theorem. Any questions up to this part? (Student speaking). There is a probability, okay.

See, we need some distribution; we need some probability distribution to tell us, that when we are going to select the next when we are going to select a move which increases the cost or decreases the cost, does it makes the cost worse? If we only select operations which improve our cost, then we will got stuck in local minima. So, we have to also choose operators which will make the cost worse once in a while. With what probability do we choose that? So, this probability of choosing will follow some distribution, right? So, the distribution that we are choosing here is this- $e^{-\delta E}$ by T.

Actually, if ΔE is negative, then it is $e^{-\Delta E / T}$. This is the distribution that we are following. I think- (Student speaking). No, see, we are always interested in improving the cost. So, if your operator is improving the cost, just go ahead and take it- (Student speaking). No, no. This is a maximization problem- the algorithm is given for a maximization problem. So, if ΔE is greater than 0, then it is improving your cost. (Student speaking). That is because **that is because that** once in a while, you will also have to take moves, which is going to make your cost worse.

If you do not do that, if you conservatively just stick to the ones which are improving your cost, then you will get stuck in local minima. (Student speaking). That time, you have to spend, because if you do just gradient descent or hill climbing, you are basically doing a greedy and you know there all problems you cannot solve with greedy. Therefore, you will have to spend more time in exploring other parts of this search space, in order to ensure that you are reaching the global minima.

That is why you have to try out those other things also, but the philosophy is that we want to do this trying out in such a way, that we are eventually guaranteed of reaching the global minima. That is what I was explaining, that you shake hard at the beginning and then slowly reduce it, so that at some point of time, you can escape out of the local minimas, but cannot escape out of the global minima. So, that is the idea. (Student speaking). How would you know that you are in a local minima?

How would you know? (Student speaking). You can- if you reach a local minima, you will find that none of your operators are improving the cost any more. So, you have exhausted all operators, and no matter which operator you apply now, your cost is not improving any more. That indicates that you are in the local minima- local or global, you do not know, right? If you start checking then, then your whole effort of coming down to this minima is lost, right? We keep that- start doing the shaking vigorously at the beginning only and then slowly bring it down, and you can think of this thing in reality. If I give you a thing and which has this kind of surface, and put a ball and then ask you to

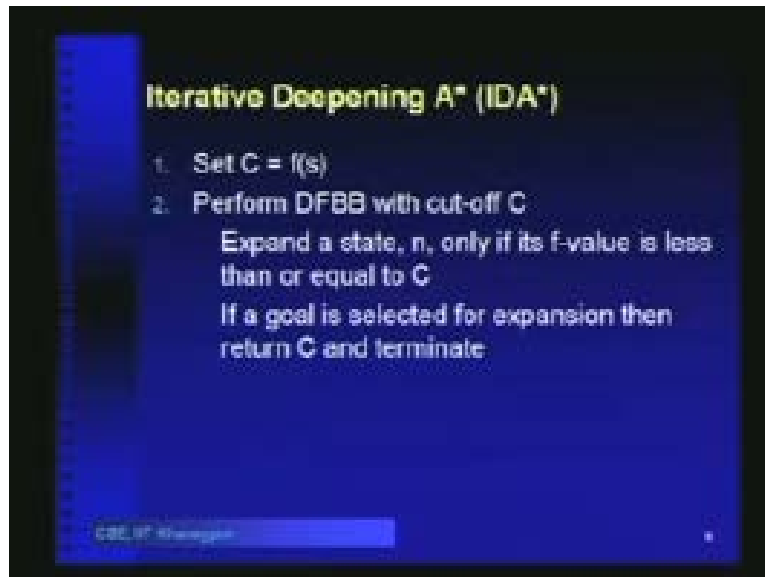
do this, you will see that it will actually go down into the global minima if you follow this, right?

This is 1 thing that we know is going to take us to the global minima and mathematically also, we have proved that it will actually go into the global minima. But if you are solving a problem for which sub-optimal solution is fine, then just go ahead and use a temperature schedule which is quite fast. You can quench it; you do not need annealing; you need annealing only when your optimization requires very accurate solution. In most problems that are very complex in nature, you will find that getting a reasonably good solution fast is not very difficult.

The difficult thing is to improve it from that point to the actual optima. But in many cases also, that is not important. Like in bio-informatics, for example, there are many cases; the exact optimization is not important because natural processes do not really care that much about the exact optima. But there are problems also, where you will fight for every inch of it. 1 example is when you are doing timing in chips- in VLSI chips- you have every inch of it is valuable; any gap anywhere, will prevent you from going into the gigahertz bit.

Now, you have these processors increasing their chip every year; there is an enormous amount of effort which goes into optimizing the time. I will ha We have We have very little time, so I will just briefly touch up on some of the memory bounded search strategies which have come up. Now, the importance of memory bounded search strategy is because search algorithms like A* are not useful in practice, because the size of open and close becomes so large that you cannot store it in main memory and that is going to really destroy the performance of the algorithm.

(Refer Slide Time: 41:09)



There are different approaches to do this. Now, we have already studied iterative deepening A*, where we progressively increase the cost bound and perform depth first branch and bound with that cut off. Now, the problem with iterative deepening A* is that it uses too less memory. What is the memory that it uses? It uses only the currently expanded path, right? And then, it just backtracks and tries another path, etc. So, **there was** at the time when this algorithm came up, this was proposed by Korf and others.

At the time when this algorithm came up, there was also the feeling that we need algorithms which, given an amount of memory, can utilize that memory and only remove parts of the memory if we need to. So, can we have a search algorithm which takes as a parameter, a given amount of memory, and then performs A* like search, on that part of the memory? What is the important issue here? Important issue is, when we run out of memory, suppose we are given a budget of m .

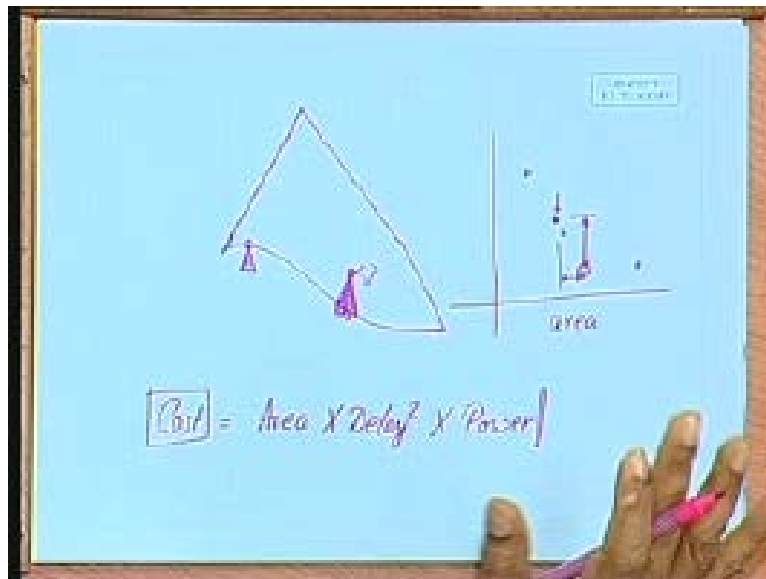
If we used up all of our memory m and we need more, then we have to remove some parts of the state space; we need to remove some part of what we have stored. Assume

that **A*** is we are running A* with open, closed, etc., and we have run out of memory. So then, there must be some part of open or close, whatever, that we will discard. So, which part will we discard? And if we discard some part, we have to make sure that when that part- now, the part that we have discard may have to be regenerated again.

Because we are discarding 1 part of the state space tree, say, suppose, we have this part; currently, we have stored this this much. And now, I want to expand a node here, so in order to create space, I will have to throw away something, right? Now, which 1 will I throw away? I will throw away those parts which have the maximum cost. Let us say that I have some node here which has the maximum cost. I throw that away; so, I throw, say, this part of the key out. My new thing that I am storing is everything except this part, and then, this part of the memory is utilized to expand this.

But then, there is a problem. See, we now have this node in the front here. We had previously expanded it, okay? Now, what may happen is that, **if you if this part** if the cost of this part again grows, then it may show up that you discard this part now and then go and expand this part, and you can go in cycles, expanding the same portions of the state space repeatedly. What we need to do is to back up some cost from here back into the node, so whatever we had seen here, pick up the most promising node on this frontier that we are discarding, and back up its cost here.

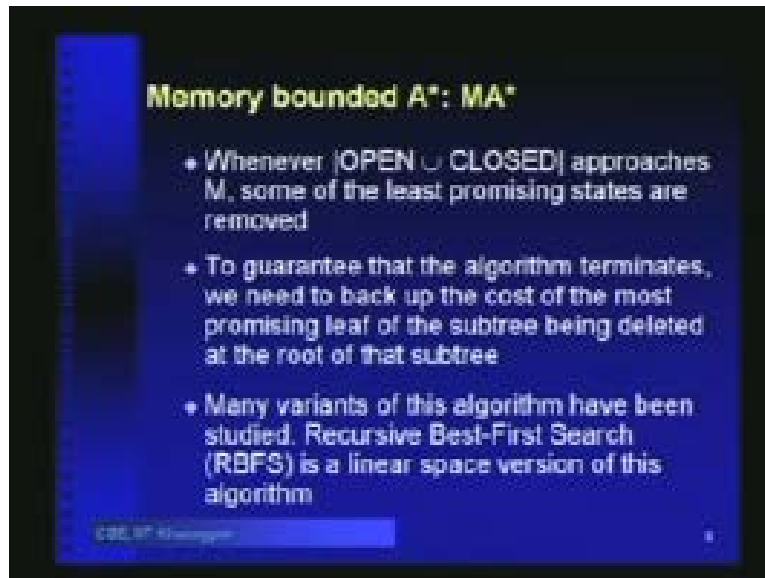
(Refer Slide Time: 49:33)



Now, so that when we are again picking this node up, then, we will at least go beyond this point, right? When we are again picking up this node, we will be picking it up on the basis of the backed up cost. So, until we get a node which has expanded a node, which has at least cost equal to the backed up cost, we are not going to stop. That is the addition that was done, right, so with that, there was actually a family of algorithms which came up. And the first of this was MA*, which was proposed by TPC in the early 90s, I think. So, the essential thing is that guarantee that the algorithm terminates; we need to back up the cost of the most promising leaf of the sub-tree being deleted at the root of the sub-tree.

And many variants of this algorithm has been studied, among which recursive best first search, again proposed by Korf, is a linear space version of this algorithm. And then, there have been many other improvements on whether we can progressively do a iterative define deepening from the frontier nodes, and what needs to back to be backed up where, etc. All that has been done, right? So, most of the existing algorithms that you will see that use variants of heuristics search, and there are, in fact, a growing number of search algorithm because of the complexity of problems that we have in bio-informatics, for example, where MA* and its variants are being used to a large extent these days.

(Refer Slide Time: 46:09)



There, all memory bounded search strategies- A* **in its** in its original form is not used in this case. Just to touch upon what is multi-objective search, so far, we have only considered cost functions which have only 1 objective, but in reality, we have more than 1 objective. For example, if you think of optimizing a circuit, then there are many parameters. You have area, you have delay, you have power, right? When you are synthesizing a circuit, you have to optimize all these parameters, right? There are 2 approaches to solving this problem; the classical approach was to combine the search criteria into a single scalar cost function.

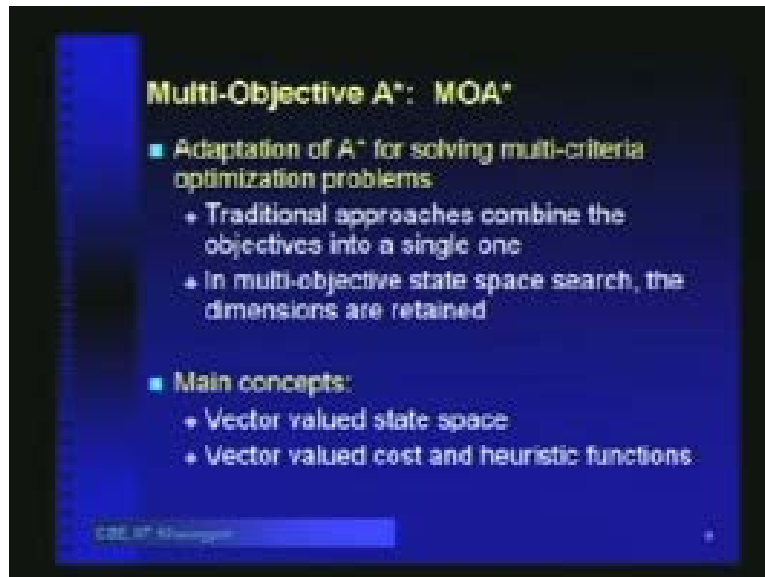
So, I say that okay, the cost that I want is area times delay square times power and I want to optimize this. This was fine at some point of time, but what is happening is that these days, we cannot combine these any more and the reason is that we do not know the nature of the solution space. If we look at the solution space as a 3 dimensional space, then you have solutions like this. Now, if you knew that **if by reducing the** if by increasing the area slightly more; suppose this is my area. If I knew that if I increase the area slightly more

from this solution to this; so, the increase in area is slight. But that is going to give me so much of improvement in delay, then, I would have chosen this solution.

But in another circuit, the sensitivity to area may not be so much. If you compromise that amount of area, then you might get a proportionate reduction in delay. Now, we do not know apriori, that what is the nature of the solution space. What people do is called design space exploration- they try out a set of good solutions, and they want to find out that what is the nature of the solution space? You want it back as a plot and then you try to see, that which 1 you want for a particular distance.

Now, how do we find these? Our objective has changed now. We have a problem where we have different dimensions- different cost functions for different dimensions- and we want to find out a set of solutions which are non-dominated. Non-dominated means that a solution is dominated by another if it is worse in all the dimensions, compared to the other. We want the non-inferior one, so it is the lower envelope of the solution space that we want. How do we use extensions of heuristics search algorithms to find out that envelope? What these search strategies do is, there it adapts algorithms like A*, MA*, IDA*, for solving multi-criteria optimization problems. Where unlike the traditional approaches, we retain the individual dimensions.

(Refer Slide Time: 51:27)



And this is done by maintaining- instead of a single scalar state space like the state vector we had, we will now have a vector which will define the state, which is the values in each of the component. And it uses vector valued cost and heuristic functions, so this is basically the idea of doing this and the search strategy also becomes slightly different because of this. And particularly interesting are game playing problems, where you have multiple dimensions, like in chess- it is very difficult to combine everything and say that this is the cost of this board position.

It is very difficult to say that. There are positional criteria, there are tactical criteria, and then the same board position- 2 grand masters will evaluate it differently, right? There, that is because they have separate set of yard sticks for measuring that. When you have this kind of structure, then you have to decide, that among this, what should be the best moves that we have to take? Can we eliminate the moves which are anyway going to be worse? So, we will not go into details of that. With these, we come to the end of the pre-mid-semester portion. After mid-semester, we will study 3 other topics- namely planning, then reasoning under uncertainty, and learning.