1. [Start of transcript. Skip to the end.](https://courses.edx.org/xblock/block-v1:ColumbiaX+CSMM.101x+3T2020+type@vertical+block@7d3957a223a5431b8c53a3aecd04cb15?show_title=0&show_bookmark_button=0#transcript-end-b1c0bf84541b4ddd99a071c58cb98e4a)
2. So, so far, we have seen all mechanisms
3. that we could use to solve a CSP.
4. Now, earlier in this lecture, I spoke
5. about the structure of the Australian map
6. and said that actually, Tasmania is separate and can
7. be considered as a separate subproblem,
8. because you could give it any color since giving it
9. any color does not violate any constraints.
10. So the idea of leveraging the problem structure
11. is to make the search more efficient
12. by looking into the graph-- the structure of the graph.
13. Again, for Tasmania, we just need
14. to solve Tasmania separately.
15. And the idea of leveraging the problem structure
16. is to find the connected component of the graph
17. constraint.
18. If we identify that this is a separate component
19. and this is a separate component,
20. we could solve them separately and then
21. gather the results of the CSP assignments
22. for each of the subproblems.
23. So in this case, we can work on independent subproblems, which
24. can reduce the complexity a lot.
25. So how does leveraging the problem structure reduce
26. the complexity of the search?
27. So first of all, suppose that we have
28. d, the size of the domain, and n, the number of variables.
29. So we call that BTS, Backtracking Search
30. is a depth first search, which means that it
31. has the same complexity.
32. It's d to the n.
33. d is a number of the size of the domain.
34. d represents the branching factors we had in search
35. in which we have the possible assignments.
36. d is the possible assignments to a variable.
37. And n is a depth of the size of the tree.
38. Now, suppose we decompose the problem
39. into subproblems with c variables per sub problem.
40. So if we do so, then we have n divided
41. by c possible subproblems, right?
42. So otherwise, if we look at the graph, constraint graph,
43. it's n divided by c possible connected components
44. in the graph.
45. So given that we have c variables per problem,
46. then it will take us a big O of d to the c.
47. This is exactly BTS.
48. But we have it now in each of the subproblems.
49. The total of all the subproblems would
50. be then equal to a big O of n divided by c, d
51. to the c in the worst case.
52. This is the new complexity when we look at the different
53. subproblem separately.
54. So let's take an example and see how this affects actually
55. the top complexity of solving a CSP.
56. So assume that we have n equals 80 variables
57. and equals 2, which means that each variable domain is
58. two values.
59. Assume we can decompose the problem
60. into four subproblems, or four connected components
61. in the constraint graph.
62. Given this decomposition, we could have about 20 variables
63. per subproblem.
64. Assume, also, that we are processing
65. 10 million nodes per second.
66. So given this number, let's check
67. what will be the complexity when we don't decompose
68. versus when we decompose.
69. So without any decomposition, the size
70. of the problem or the number of nodes would be 2 to the 80.
71. This is the complexity of DFS.
72. That's going to be d to the n.
73. That's going to be 2 to the 80, which represents 1.2 times 10
74. to the 24 possible nodes.
75. We divide this by 10 million nodes per second
76. to get an estimation of the time.
77. And this is going to be 3.8 million years.
78. Yes.
79. So with decomposition, if we check the decomposition
80. to four cognitive components, we have a complexity of what?
81. Of the number of components-- that's 4--
82. times d to the c. c is the number of variables
83. for each of the sub-components.
84. That's going to be 4 times 2 the 20,
85. which is 4.2 times 10 to the 6 nodes.
86. We're going to divide that by 10 million again.
87. And we get a complexity in time of 0.4 seconds,
88. much, much, much, less.
89. So I can see you behind a screen saying, wow, yes.
90. And to use the words from Rosslyn Norvig,
91. we will choose the problem.
92. They say that we will choose the problem
93. from the lifetime of the universe
94. to actually less than a second or even half a second.
95. This is absolutely amazing.
96. OK, so this is the take-home message
97. here-- is that if you can, check your problem structure
98. and see if there is any correct component in it.
99. You can turn the problem into subproblems.
100. And chances that you are going to improve
101. the efficiency a lot.
102. While this is very compelling, turning a problem
103. to subproblems is not always possible.
104. It can happen, but it is rare enough.
105. So in this case, we wonder whether we
106. could leverage all the structure in the problem
107. to make the search more efficient.
108. And the answer is yes,
109. it's possible.
110. If the graph has some tree structure or even nearly tree
111. structure, then we could leverage that
112. in order to do a better search.
113. We call a graph a tree if actually any two variables,
114. if you take any two variables and you
115. will find them connected by only one path.
116. And the idea is to use an extended form
117. of arc-consistency, called Directed Arc-Consistency
118. because they're going to see the graph
119. search as an oriented graph that's a tree.
120. OK?
121. So we use DAC for Directed Arc-Consistency.
122. And the CSP is called DAC consistent,
123. or Directed Arc-Consistent.
124. And there's some ordering.
125. Suppose we have x1, x2, xn
126. that represent ordering in the tree from parents to children.
127. If and only if, if you take any xi, for every xi,
128. xi is arc-consistent with each xj for which we
129. have j bigger than i.
130. OK?
131. So let's take an example.
132. So suppose we have this structure of the graph.
133. The graph constraint has--
134. we don't have necessarily roots.
135. We are going to pick one.
136. Suppose we take A as the root of the tree.
137. So if we do so, then we are going
138. to have the tree going from the root A to B. B goes to C or D.
139. And D goes to E of F.
140. So we are going to write this as this form of
141. tree-- probably, you will recognize better
142. this form of the tree.
143. A goes to B. B goes to C or D. And D goes to E
144. or F. So recognize this is a structure.
145. This is the graph.
146. So we have one, two, three, four, five, six, nodes.
147. And we have one, two, three, four, five, five edges.
148. So we have n nodes, n minus 1 edges.
149. OK?
150. So we're going to first pick a variable to be the root.
151. Then we do what we call a topological sorting
152. of the particle sort of what?--
153. means choose an ordering of the variables, such
154. that each variable appears after its parent in the tree.
155. So the variable B will be sorted after A, et cetera.
156. OK?
157. So we're going to put that ordering.
158. Each variable appears after its parent in the tree.
159. OK.
160. So for n nodes, we have n edges, just as in the example.
161. n nodes, we have specifically n minus 1 edges.
162. So we're going to make the tree directed arc-consistent or DAC.
163. We take a Big O of n.
164. Right?
165. Each consistency we take up to O d squared,
166. because you compare d possible values for d variables, again.
167. This CSP then can be solved in the linear time, which is
168. we combine these two together.
169. It's going to be a big O of n d squared, which
170. is also very good.
171. We are going to have now a linear complexity
172. in n of this tree search, rather than a graph search.
173. Now, what if the CSP cannot be represented as a tree structure
174. CSP?
175. We're going to check whether we could
176. do something that is nearly a tree structure at CSP,
177. hence, this title here.
178. So we're going to try to do that by, for example, assigning
179. one or many of the variables in the CSP with specific values.
180. So for example, if we assign sa to some color,
181. we're going to remove all these edges here
182. that link sa to the rest of the constraint graph.
183. Otherwise, it's not possible to write the tree based
184. on this graph here on the left.
185. Just because we have loops and it's not
186. possible to transform that or translate that into a tree.
187. So suppose we are going to assign a value to a variable
188. or too many variables.
189. We are going to prune all the neighbors, which will turn out
190. actually to be this new configuration here that we
191. could represent as a tree.
192. OK?
193. So we're going to transform a nearly tree-structured CSP
194. into a tree-structured CSP by just fixing some values.
195. Sometimes the benefit is absolutely huge,
196. given that the complexity could go really wide if we
197. don't do these kinds of tricks.
198. But if you want to use more tricks,
199. so these tricks actually turn the graph into a tree.
200. But there are lots of other tricks
201. that we could use in CSP.
202. So have fun if you want to explore
203. more of these kinds of tricks to play
204. with the structure of the graph.
205. Let's now summarize.
206. We have the CSPs, an important class of problems
207. in artificial intelligence.
208. And CSPs are a special kind of search problems
209. in which we have the state defined
210. by the values of a fixed set of variables.
211. And the goal test is actually defined by the constraints
212. on variable values.
213. We want to be able to assign values to the variables
214. without violating the set of constraints that are part
215. of the definition of a CSP.
216. Typically, we use what you call a backtracking search.
217. It's a depth-first search with one variable assigned
218. at a time.
219. We use heuristics or tricks to help the backtracking search.
220. This could be, for example, using variable ordering and value
221. selection heuristics.
222. So this can help, actually, do a better backtracking search.
223. We also saw that forward checking
224. prevents assignments that guarantee later failure.
225. So if we are going some path in the search
226. that will lead to failure,
227. we need to just cut it earlier and not go to that path.
228. We also saw that in CSPs we could
229. do two things, search or inference or both
230. of them intertwined.
231. So in inference, we talk about constraint propagation,
232. propagating the information among the unassigned variables
233. and assigned variables.
234. And this is typically done to what we call arc-consistency.
235. It is an important mechanism in CSPs
236. that does some additional work to constraint values
237. and detect inconsistencies.
238. And sometimes, this is enough to solve entire CSPs.
239. We also saw that leveraging the problem structure
240. can be very important and can lead
241. to a big gain in efficiency.
242. For example, if a CSP has a tree structure,
243. it can be solved in a linear time.
244. Sometimes the CSP is nearly tree-structured.
245. And we could transform it into a tree structure CSP.
246. So further exploration, if you want
247. to go deeper into the topic, you could check,
248. for example, local search.
249. So we have seen local search in the search algorithms.
250. How can local search be applied or used to solve CSPs?
251. That's another successful branch in solving
252. CSPs using local search algorithms,
253. such as hill climbing and genetic algorithms.
254. An important point to consider here is that CSPs are powerful.
255. And they are powerful because they are domain-independent,
256. which means that if you have a problem to solve using CSP,
257. you could use any CSP solver, as far as you formalize
258. the problem specifically.
259. If you are specifically defining your variables, your domains,
260. and your constraints, you could plug that into any CSP solver.
261. If you want to play with some solvers,
262. I recommend this one from the AIspace.
263. That's a very good example.
264. But there are many others out there.
265. Finally, I would like to dedicate this lecture
266. to David Waltz, who was a friend, a colleague,
267. and my boss between 2006 and 2012.
268. David Waltz was a computer scientist
269. who made significant contributions
270. to the field of artificial intelligence,
271. including constraint satisfaction
272. problems, case-based reasoning, and the applications
273. of massively parallel computations to AI problems.
274. He held several positions in both academia and industry.
275. And at the time of his death, he was a Professor
276. of Computer Science at Columbia University, leading
277. the Center for Computational Learning Systems
278. here at Columbia.
279. Remarkably, his PhD dissertation on "Computer Vision"
280. initiated and invented the field of constraint propagation,
281. which allowed a computer program to generate
282. a detailed three-dimensional view of an object, given
283. a two-dimensional drawing with shadows.
284. So this lecture is for you, Dave.
285. Thank you very much for your attention.
286. I hope you learned a lot from this CSP lecture.
287. Looking forward to seeing you in the next one.