Handling items that depend on each other

Suppose you want to go to Paris, so you add a couple of items on the list.

EIFFEL TOWER	1½ DAY	8
THE LOUVRE	1½ DAY	9
NOTRE DAME	1½ DAY	7

These places take a lot of time, because first you have to travel from London to Paris. That takes half a day. If you want to do all three items, it will take four and a half days.

Wait, that's not right. You don't have to travel to Paris for each item. Once you're in Paris, each item should only take a day. So it should be one day per item + half a day of travel = 3.5 days, not 4.5 days.

If you put the Eiffel Tower in your knapsack, then the Louvre becomes "cheaper"—it will only cost you a day instead of 1.5 days. How do you model this in dynamic programming?

You can't. Dynamic programming is powerful because it can solve subproblems and use those answers to solve the big problem. *Dynamic programming only works when each subproblem is discrete—when it doesn't depend on other subproblems.* That means there's no way to account for Paris using the dynamic-programming algorithm.

Is it possible that the solution will require more than two sub-knapsacks?

It's possible that the best solution involves stealing more than two items. The way the algorithm is set up, you're combining two knapsacks at most—you'll never have more than two sub-knapsacks. But it's possible for those sub-knapsacks to have their own sub-knapsacks.

IT'S NOT POSSIBLE TO HAVE THREE SUB-KNAPSACKS
BUT IT IS POSSIBLE TO HAVE SUB-KNAPSACKS THAT HAVE THEIR OWN SUB-KNAPSACKS.