1.

Dear Tom,  
  
Thank you for writing, and for your fun question.  
  
It is indeed possible to model your problem as a so-called:  
  
"finite-horizon  MDP with expected total reward objective"  
  
One can then, in principle, solve such a model to find  
the optimal policy by using a "backward induction" dynamic programming  
algorithm.  
  
However, because of the finite-horizon structure,  
and because both the amount of money accumulated  
so far, and the number of elves obtained so far, are part of the state, and because there are  
ponentially many (but only a bounded number of)  
choices for "moves" at each state,  
one important issue is that the number of states and moves in the model can potentially get very large  
(this is often referred to as the "curse of dimensionality"  
 or the "state explosion problem").  
  
  
(But the number of states/moves is nevertheless always finite,  
 because of the finite-horizon nature (namely a fixed number, 10, of  
 rounds) ensures that the total amount of elves and money accumulated  
 at any time during play, and upon termination after 10 rounds,  
 can not exceed a specific finite amount.)  
  
Because of the potentially large state space, such a model will not be very easy to solve by hand: you will need a computer implementation of, e.g., the backward induction algorithm to solve it.  
This is so especially if the number of rounds  
and initial number of elves/money is large.  In your case, with 10 rounds, and with initially 12 elves and no money, it should hopefully be managable for a computer to solve such  
a model,  
(with, say, a 10^6 rounds, and 10^5 initial elves,  
 it would become prohibitively inefficient  
 to solve it by backward induction, even on a computer,  
 and one would need to try to invent new specialized ways  
 to solve this specific model).  
  
  
In more detail, here is a way that you could set this  
up as an MDP model.   We view things from the point of  
view of a \*single\* player who is trying to optimize  
its expected total earnings during the 10 rounds.  
  
1) each "state" of the MDP encodes:  
   the amount of  money accumulated thus far by the player,  
   and the number of elves that the player has,  
   the number of "rounds" that have  
   been played thusfar,  
   and finally, whether or not the player has  
   already made its choice regarding how  
   many elves to buy before the next round (and after the prior one).  
  
   So, a "state" is given by a quadruple of numbers: (i,j,k,b),  
   where i is a number between 0 and 10, specifying  
   the number of rounds that have been played thusfar,  where j  
   specifies the number of elves the player has,  
   where k specifies the amount of money the player has,  
   and where b is a "bit" (i.e., either 0 or 1) which  
   specifies whether the player has already made its  
   choice regarding whether to purchase any elves prior  
   to the next round  (b=0 means the choice has not yet  
   yet been made, and b=1 means that the choice has been made).  
  
   So the "initial state" that the player  
   starts in is: (i,j,k,b) := (0,12,0,0)  
  
  
2) now lets consider what happens in any state.  
  
   When the player is currently in a state (i,j,k,0),  
   i.e., it has not yet made a choice about  
   its purchase of elves before the next round,  
   the first thing it needs to do is to choose  
   how many elves it want to buy with its current money  
   k.   Of course, it can only buy at most  
   \floor(k/75) states  
   (where floor(x) means the larges integer less than the  
    real or rational number x).  
  
   So, the player has precisely  \floor(k/75)+1 choices:  
   it can choose to buy 0,1,2, ...., or \floor(k/75) elves  
   before the next round.  
  
   Suppose the player makes its choice, and decides to  
   buy  m elves,  
   then the new "state" after this choice "move" will be:  
  
    (i,j+m,k-(m\*75),1)  
  
   Next, we have to specify what happens when  
   a  player is in some state of the form  
  
   (i,j,k,1)  
  
   (in other words, after it has chosen its elve purchase).  
  
   In such a state, the player next has to choose how many elves  
   to send into the woods, into the forest, and into the mountains,  
   respectively.  
  
   Specifically, it has j elves to use up,  
  
   and it has to make a "choice move"  specified by a  
   triple of non-negative numbers (a,b,c), such that  
  
    a + b + c <= j,  
  
   where a is number of elves that it chooses to send to the woods,  
         b is the number elves that it chooses to send to the forest,  
         c is the number that it chooses to send to the mountains.  
  
  
   Having made that choice (a,b,c),  then with certain  
   probabilities the player accumulates some payoffs  
   from the trees cut by the elves, and with certain  
   probabilities loses some elves, etc.  
  
   it is possible to calculate these probabilities  
   in such a way as so set up  
   "probabilitic transitions" such that after  
   choice (a,b,c) is made,  the "state" transitions  
   with some specific probability p(a,b,c,alpha),  
   to a new state alpha.  
  
   by this I mean the following:  
  
   if currently in state (i,j,k,1) the player makes choice  
   move (a,b,c), then for some calculatable  
   probability p(a,b,c,j',k')  
  
   (which we can calculate based on the 1/3 probability  
    of bad weather, and the choices (a,b,c) of where  
    to send the elves in that round)  
  
   we transition next with probability p(a,b,c,j',k') to the state  
  
   (i+1,j-j',k+k',0)  
  
    here j' is the number of elves  
    lost in the mountains, which can be between  
    0 to c,  and k' is the amount of money made by the  
    elves in that round.  
  
    Note that the number of possible values of j' and k'  
    is bounded, so this remains a "finite state" model.  
  
   (Notice that we have in the above state updated the number of  
    rounds played thusfar to i+1, and we have reset the bit b=0  
    to indicate that we have not yet chosen  
    whether to purchase elves before the (new) next round).  
  
  
I will not specify in detail how to calculate the above  
transition probabilities, but if you know a little  
basic probability it is not difficult see how one  
can calculate them.  
  
  
In this way, the player continues to play until  
we have finished 10 rounds, i.e., until we have  
reached a state (10,j,k,0).  
  
In such a state, the play terminates, and the player  
receives the "payoff"  k  (which is the amount of  
money it has accumulated by the end).  
  
Of course, the player's goal in this model is  
to maximize the EXPECTED VALUE  
of its total payoff at the end of play (i.e., upon termination  
after 10 rounds), starting from the initial state  
(0,12,0,0).  
  
That completes the description of the finite-horizon  
MDP.   (This is in fact also an example of  
what one calls a 1-player  
finite extensive form game of perfect information,  
with chance moves.)  
  
Once we "build" such a model, we can "solve" for the  
optimal strategy by a "backward induction" algorithm.  
  
I will not explain this algorithm in detail here  
(it is described more generally in the lecture notes  
 for my algorithmic game theory course), but the  
basic idea is that one starts at the "leaves" of  
the game tree (i.e., at the terminal states in  
this MDP), and by induction going back up the tree,  
one makes an optimal choice for (a,b,c)  
at that state, depending on the already computed  
optimal expected payoffs that have been calculated  
for all of its "children" (i.e., states in the "next"  
round).  
  
  
I hope the above information is helpful to you.  
  
If you can not understand what I have said, then if you wish to visit Edinburgh some day (when and if we both find time) I could try to explain it to you and your students in person.  
  
  
Best regards,  
Kousha

2.

Hi Tom,

Do you have to generate all nodes in the game, starting from (0,12,0,0), and  
then select the leaves from which to start? Otherwise, how do you generate  
(all) possible leaves? From reading about DP, is it correct that generating  
all nodes, and then working from the bottom up, is called tabulation?

You don't explicitly need to generate all possible leaves directly.  
  
Instead, one simple way to think about how do  
dynamic programming like this it is this  
(this is called the "memoization" approach,  
 with recursion,  
 that I referred to in my prior email):  
  
You essentially need to write a recursive function,  
  
OPT(i,j,k,b)  
  
which takes as input a "state"  (i,j,k,b),   and  
returns both the optimal expected total payoff starting  
from that state, as well as the optimal "first move",  
from that state, i.e., an optimal triple (a,b,c)  
which you should choose for the number of elves  
to send to each of the three possible places.  
  
Of course to get the "answer" for the original  
question you will call this function with (0,12,0,0).  
  
OPT(i,j,k,b) itself can be defined as a  
function which recursively calls itself  
for new states (i',j',k',b') as needed,  
but such that, \*before\* doing a recursive  
call on (i',j',k',b'),   we CHECK in  
a lookup table whether the value/action  
solution for (i',j',k',b') has already  
been computed before (i.e., "memoized"),  
in which case we do not need to  
make the recursive call, and can  
simply use that answer.  
  
Moreover, once we do compute an answer  
during the computation for some state  
(i',j',k',d'),  we insert that answer  
into the lookup table for future use.  
  
(The lookup table can be an efficient  
 hash table, or array, or other efficient  
 data structure suitable for this purpose.)  
  
Now, of course the key question is  
"what is the nature of the recursive procedure  
that defines OPT(i,j,k,b)".  
  
This simply follows the definition of the  
(partially probabilistic) "game tree" that I described in my prior  
email.  
  
I can give more details, but at some point this will amount  
to writing out all the details of the dynamic programming algorithm  
for you.  
I hope the above hints give you enough to work on to write such  
a recursive procedure yourself.  
  
  
Best,  
Kousha

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif