20191105

Branch and Bound Continued

Let's think about how we can come up with the l_P and u_P bounds for the partial solutions.

Remember that l_P has to be a valid lower bound on the cost of the best solution that extends P. The most common approach to finding l_P is to focus on costs that are shared by all solutions that extend P. We typically separate these costs into two parts:

Cost So Far (CSF) : each decision we have made to construct this partial solution P has some cost associated with it. The sum of these is the Cost So Far. All solutions that extend P will also include these costs.

Guaranteed Future Costs (GFC) : each decision yet to be made may have a minimum cost associated with it. The sum of these is the Guaranteed Future Cost. All solutions that extend P will cost at least this much for the decisions that extend P.

We use $l_P = CSF(P) + GFC(P)$

Now u_P has to be a valid upper bound on the cost of the best solution that extends P. The most common approach to finding u_P is to observe that by definition, the cost of the best solution that extends P is $\leq$ the cost of every solution that extends P. So if we randomly (or cleverly) construct a solution S that extends P, the cost of S is a valid u_P. As we did for l_P, we usually break the cost into two parts:

Cost So Far (CSF) : same as above. These are costs that are shared by all solutions that extend P.

Feasible Future Cost (FFC) : we extend P to a full solution S. The costs incurred by the decisions that we make during the extension process are summed to give the Feasible Future Cost.

We use $u_P = CSF(P) + FFC(P)$

We can think of $u_P - l_P$ as a measure of the ambiguity of P. If l_P and u_P are far apart we have very little knowledge about the best solution that extends P. On the other hand if l_P

and u_P are close together then we have a very good estimate of the cost of the best solution that extends P.

Also, the closer together we can bring l_P and u_P, the less likely it is that different partial solutions will have overlapping bounds. And if the bounds for two different partial solutions do not overlap, then we can throw one of them away!

Example: Suppose we have two partial solutions P_1 with bounds [7, 15] and P_2 with bounds [12, 20]. We can't be sure whether the best solution that extends P_1 costs less than the best solution that extends P_2, or vice versa. We have to keep both of them.

But if P_1 has bounds [7, 10] and P_2 has bounds [12, 20], we can see that the best solution that extends P_2 ***cannot*** be better than the best solution that extends P_1. So we can throw P_2 away. In fact for any two partial solutions P_1 and P_2, if $u_{P_1} \leq l_{P_2}$ then we can throw away P_2

Throwing away partial solutions is what B&B is all about. Typically the number of partial solutions and potential full solutions is exponentially large; we want to keep the number of partial and full solutions that we actually deal with as small as we can. We do this by using our calculated bounds to eliminate partial solutions that cannot lead to an optimal solution.

Here's a demonstration of the Branch and Bound method.

Suppose we have n persons and n tasks – each person is to be assigned to one of the tasks. Because the people have different skill sets, the time required to complete the tasks can be different for each person. Our goal is to assign the tasks in such a way that the total time required is minimized.

For example, suppose we have 5 persons $\{P_1, P_2, P_3, P_4, P_5\}$ and 5 tasks $\{T1, T2, T3, T4, T5\}$. We can represent the time requirements with a matrix:

	T1	T2	T3	T4	T5
P1	10	15	4	12	9
P2	7	18	3	10	16
P3	4	5	14	12	10
P4	17	2	18	6	21
P5	21	18	2	10	25

At this point I have to admit that there exists a polynomial-time algorithm to solve this problem! It is called the Hungarian Method – I hope to cover this algorithm at the end of this course because it is a brilliant and widely applied algorithm.

However it's worth looking at how we can solve this problem using Branch and Bound because it introduces some clever techniques that we can apply to other problems that do not have polynomial-time algorithms.

Step by step:

Objective function: we are trying to minimize the total time – ie the sum of the times required for the assignments we choose.

Decision sequence: Our i^{th} decision will be to assign a task to the i^{th} person. Note that we could turn this around and make our i^{th} decision to assign a person to the i^{th} task.

Initial Global Upper Bound:

To find this initial bound we can simply generate a solution and use its total cost – the optimal solution's cost must be $\leq$ this.

One way to do this is to assign T1 to P1, T2 to P2 etc. In the example above this gives a total of 73. This is a very quick and easy way to get an initial upper bound.

If we are willing to do a bit more work we can improve this (remember, we always want to make our upper bounds lower and our lower bounds higher). We can apply a simple greedy heuristic: give person P1 the task they can do the fastest, then give P2 the remaining task they can do the fastest, etc. This results in this assignment:

	T1	T2	T3	T4	T5
P1	10	15	4	12	9
P2	7	18	3	10	16
P3	4	5	14	12	10
P4	17	2	18	6	21
P5	21	18	2	10	25

This has a total cost of 47 – much better than the 73 we had before. The trade-off is that it takes $O(n)$ time to get the trivial bound, and $O(n^2)$ to get the improved bound … but if this results in a significant speed-up of the Branch and Bound algorithm it may be worth it.

There are more things we can try – for example, we could apply the Greedy heuristic n times, starting with each of the persons. This raises the time requirement to $O(n^3)$ … but it may result in a very good initial upper bound.

Now our algorithm proceeds in the manner explained previously. We generate all the partial solutions that result from our options for the first decision. Since our first decision is to choose a task for P1, there are 5 options.

We need some notation for partial solutions. Since each consists of task assignments to the 5 people in order, as well as a lower and an upper bound, we can use notation like this:

- a vector of length 5 in ( ) brackets showing the task assignments, with "-" for assignments not yet made, and
- l_P and u_P in [ ] brackets

Thus (3 5 - - -) [37 91] represents the partial solution "Assign T3 to P1, Assign T5 to P2 $l_P = 37, u_P = 91$"

Using this notation, our first initial partial solution is

(1 - - - -) [? ?]

But what are the lower and upper bounds? It's easy to see that CSF = 10. We can also see that with T1 assigned to P1, P2 is going to cost at least 3, P3 is going to cost at least 5, and P4 and P5 are each going to cost at least 2. So GFC = 3+5+2+2 = 12. This gives l_P = CSF + GFC = 22

For FFC we can apply the same Greedy Heuristic as we did to get U_{Global}.
This yields FFC = 39, so u_P = CSF + FFC = 49

Thus our first partial solution is (1 - - - -) [22 49]

The rest of our partial solutions are (assuming my arithmetic is correct – I think I got different values when I did this in class!)

(2 - - - -) [30 53]
(3 - - - -) [27 47]
(4 - - - -) [23 46] $\longleftarrow$ This reduces our U_{Global} to 46
(5 - - - -) [20 28] $\longleftarrow$ This reduces our U_{Global} to 28 !!

Since U_{Global} = 28, we can throw away the partial solution (2 - - - -) [30 53]

Our set of "live" partial solutions is thus

(1 - - - -) [22 49]

(3 - - - -) [27 47]

(4 - - - -) [23 46]

(5 - - - -) [20 28]

We choose the one with the lowest l_P, which happens to be the last one, and we expand it by trying all the choices for the next decision. Since T5 has been assigned, the choices for P2 are T1, T2, T3 and T4

At this point the matrix is reduced to this:

	T1	T2	T3	T4	
P2	7	18	3	10	
P3	4	5	14	12	
P4	17	2	18	6	
P5	21	18	2	10	

The four new partial solutions are (with some of the arithmetic shown)

(5 1 - - -) [9+7+9 9+7+13] .... (5 1 - - -) [25 29]

(5 2 - - -) [9+18+12 9+18+12] .... (5 2 - - -) [39 39] $\longleftarrow$ Throw away!

(5 3 - - -) [9+3+8 9+3+16] .... (5 3 - - -) [20 28]

(5 4 - - -) [9+10+8 9+10+8] .... (5 4 - - -) [27 27] $\longleftarrow$ This reduces our U_{Global} to 27 !!

so our complete set of live partial solutions is

(1 - - - -) [22 49]

(3 - - - -) [27 47] $\longleftarrow$ Throw away – it can't beat the last one!

(4 - - - -) [23 46]

(5 1 - - -) [25 29]

(5 3 - - -) [20 28]

(5 4 - - -) [27 27]

The best of these (lowest l_P) is (5 3 - - -) [20 28] so we expand this. Our next decision is to assign a task to P3 and the choices are T1, T2 and T4. The matrix is now reduced to this

	T1	T2		T4	
P3	4	5		12	
P4	17	2		6	
P5	21	18		10	

The new partial solutions are

(5 3 1 - -)[12+4+12 12+4+12] …. (5 3 1 - -)[28 28] $\longleftarrow$ Throw away!
(5 3 2 - -)[12+5+16 12+5+27] …. (5 3 2 - -)[33 44] $\longleftarrow$ Throw away!
(5 3 4 - -)[12+12+20 12+12+23] …. (5 3 4 - -)[44 47] $\longleftarrow$ Throw away!

So we are left with

(1 - - - -) [22 49]

(4 - - - -) [23 46]

(5 1 - - -) [25 29]
(5 4 - - -) [27 27]

We choose (1 - - - -) [22 49] to expand. The matrix for this partial solution looks like

		T2	T3	T4	T5
P2		18	3	10	16
P3		5	14	12	10
P4		2	18	6	21
P5		18	2	10	25

And we proceed by generating the four new partial solutions

(1 2 - - -) [46 46]	$\longleftarrow$ Throw Away
(1 3 - - -) [30 49]	$\longleftarrow$ Throw Away
(1 4 - - -) [29 68]	$\longleftarrow$ Throw Away
(1 5 - - -) [35 39]	$\longleftarrow$ Throw Away

See how effective this method is for eliminating branches! For the second time, just by comparing the l_P values to U_{Global} we can close off an entire branch of the decision tree.

The set of live partial solutions is now

(4 - - - -) [23 46]
(5 1 - - -) [25 29]
(5 4 - - -) [27 27]
(5 3 1 - -)[28 28]

We choose (4 - - - -) [23 46] to expand. The four new partial solutions are

(4 1 - - -) [ ]
(4 2 - - -) [ ]
(4 3 - - -) [ ]
(4 5 - - -) [ ]

I'll let you compute the l_P and u_P for each of these. Remember to include the CSF values.

The algorithm rumbles on to its inevitable conclusion. Up to this point we have generated 20 partial solutions. Finishing off the algorithm might take 20 more (probably fewer!) Compare this to exploring the entire set of complete solutions, which has 5! = 120 members.

But here comes the main reason for introducing this problem: there's a technique we can apply to extract "hidden" information from the matrix that can help us with our bounds. Notice that person P1 takes at least 4 hours for each task. We can subtract 4 from every value in P1's row without changing the optimal task assignment – we just have to remember to add the 4 back in to get the actual cost. And … we can do the same for every person: we subtract their smallest time requirement from all the values in their row. This gives this matrix:

	T1	T2	T3	T4	T5
P1	6	9	0	8	5
P2	4	15	0	7	13
P3	0	1	10	8	6
P4	15	0	16	4	19
P5	19	16	0	8	23

The total extracted costs are 4+3+4+2+2 = 15

But we're not done yet: Notice that all the values in the column for T4 are positive. In fact we can see that no matter who does task T4 it will take at least 4 hours. So we can subtract 4 from every value in this column and add 4 to our accumulated costs … and by the same logic we can extract 5 from the column for T5.

This gives

	T1	T2	T3	T4	T5
P1	6	9	0	4	0
P2	4	15	0	3	8
P3	0	1	10	4	1
P4	15	0	16	0	14
P5	19	16	0	4	18

The total extracted cost is now 15 + 4 + 5 = 24

Very clever, but what's the point? Look what happens when we apply the Greedy heuristic to get an initial upper bound:

	T1	T2	T3	T4	T5
A	6	9	0	4	0
B	4	15	0	3	8
C	0	1	10	4	1
D	15	0	16	0	14
E	19	16	0	4	18

0+3+0+0+18 = 21

When we add the extracted costs we get 21+24 = 45 as the cost of this solution … and that is LOWER than the total cost we got by just applying the Greedy heuristic to the original matrix. So we have found a better initial value of U_{Global}.

(Note that if we try starting the Greedy heuristic with B, C, D or E we get even better results.)

Now we turn to computing the l_P and u_P values for partial solutions. As before we will let $l_P = CSF(P) + GFC(P)$ and $u_P = CSF(P) + FFC(P)$

CSF : just the sum of the costs of the decisions made so far, plus the extracted costs as defined above.

GFC: pretty much the same as before, but we can pull the reduction trick again. If there is any row in the reduced matrix in which all the values are positive, we can subtract the smallest value from every value in the row. Then we add the extracted value to the accumulated extracted costs for this partial solution. And we can do the same for each column! As we proceed, more and more elements of the matrix are reduced to 0, and that has a benefit as we shall see in a moment.

FFC: For the feasible future cost we can use the same approach as we did for the initial upper bound: apply the Greedy heuristic. Now think about those 0's that are starting to crop up in the matrix. The more 0's there are in the matrix, the more likely we are to find a feasible solution with a total future cost of 0. And since that obviously cannot be improved on, it means we have found the best possible expansion of the current partial solution. So we can replace this partial solution with the full expansion we just found. This accelerates the algorithm by eliminating all the iterations required to explore other possible expansions of this partial solution.

This will probably make more sense if we return to our example. Consider the first step of the algorithm. Here's the matrix again, with the reduction step applied – remember this gave us an extracted cost of 24 that we will include in the CSF of every partial solution (actually since it contributes equally to ***all*** solutions, we could just forget it – but at some point we will have to add it in to get the true cost of the optimal solution. We might as well include it right from the start).

Remember that in this version our $U_{Global} = 45$

	T1	T2	T3	T4	T5
P1	6	9	0	4	0
P2	4	15	0	3	8
P3	0	1	10	4	1
P4	15	0	16	0	14
P5	19	16	0	4	18

Our first partial solution (1 - - - -) assigns T1 to P1. This pair now has a cost of 6. Thus the CSF for this partial solution is 24 + 6 = 30. This assignment reduces the matrix to this:

	T2	T3	T4	T5
P2	15	0	3	8
P3	1	10	4	1
P4	0	16	0	14
P5	16	0	4	18

The row for P3 has all values ≥ 1 so we can extract a cost of 1 from this row, giving

	T2	T3	T4	T5
P2	15	0	3	8
P3	0	9	3	0
P4	0	16	0	14
P5	16	0	4	18

We can consider the 1 we extracted as the GFC, but in practical terms it is easier to recognize it as part of the CSF since it results from the decisions made up to this point. Thus we now have CSF = 30 + 1 = 31

Now every row and column contains a 0 so no further reductions are possible at this time.

Having made this reduction, GFC = 0. In fact, using this method GFC will ***always*** = 0 since we make sure there is a 0 in every row. We end up with l_P = 31 for this partial solution.

For the FFC we apply the Greedy heuristic. We get a FFC of 18. This gives u_P = 31 + 18 = 49

So this partial solution looks like (1 - - - -) [31 49] - note that this is a narrower range than we had with the original version of the algorithm

The next partial solution is (2 - - - -). This has a cost of 9, and the reduced matrix is

	T1	T3	T4	T5
P2	4	0	3	8
P3	0	10	4	1
P4	15	16	0	14
P5	19	0	4	18

In this matrix the only reduction we can make is extracting 1 from the T5 column, which gives

	T1	T3	T4	T5
P2	4	0	3	7
P3	0	10	4	0
P4	15	16	0	13
P5	19	0	4	17

CSF = 24 + 9 + 1 = 33
GFC = 0
FFC = 17

So this partial solution is (2 - - - -) [33 50]

Now let's look at the partial solution resulting from assigning T3 to P1: (3 - - - -)

This decision has a direct cost of 0. The reduced matrix is

	T1	T2	T4	T5
P2	4	15	3	8
P3	0	1	4	1
P4	15	0	0	14
P5	19	16	4	18

We can reduce the row for P2 by 3, the row for P5 by 4, and the column for T5 by 1. We have extracted another 8 units of cost, and the resulting matrix is

	T1	T2	T4	T5
B	1	12	0	4
C	0	1	4	0
D	15	0	0	13
E	15	12	0	13

CSF = 24 + 0 + 8 = 32

GFC = 0

FFC = 13

l_P = 32

u_P = 45

So this partial solution is (3 - - - -) [32 45]

The next partial solution is (4 - - - -). This assignment has an immediate cost of 4, and reduces the matrix to

	T1	T2	T3	T5
P2	4	15	0	8
P3	0	1	10	1
P4	15	0	16	14
P5	19	16	0	18

Here again we can reduce the T5 column by 1, giving

	T1	T2	T3	T5
P2	4	15	0	7
P3	0	1	10	0
P4	15	0	16	13
P5	19	16	0	17

CSF = 24+4+1 = 29
GFC = 0
FFC = 17

l_P = 29
u_P = 46

This partial solution is (4 - - - -) [29 46]

The last of our initial partial solutions assigns P1 to T5. This has an immediate cost of 0, and reduces the matrix to

	T1	T2	T3	T4
P2	4	15	0	3
P3	0	1	10	4
P4	15	0	16	0
P5	19	16	0	4

No reduction is possible on this matrix because there is at least one 0 in each row and each column.

CSF = 24 + 0 + 0
GFC = 0
FFC = 4

l_P = 24
u_P = 28 $\longleftarrow$ Yeeee hah!

The effect of this u_P is to reduce U_{Global} from 45 to 28 – this is what happened in the original version of the algorithm too. But look what happens in this version: because our new l_P values for the other partial solutions are ***all*** > 28, we can **throw them all away!!!**

The only partial solution we keep is the last one :

(5 - - - -) [24 28]

This is as good as it gets. With a bit of careful analysis we were able to eliminate all but one of the options for the first decision. In other words we know the optimal solution starts with assigning T5 to P1. We can now move on to analyzing the choices for P2, then P3 etc. At each stage we do whatever reductions we can and eliminate as many partial solutions as we can.

And so we carry on in standard B&B fashion until we find an optimal solution. The only tricky bit is making sure we keep track of the extracted costs properly, which is actually pretty easy: each partial solution inherits all the extracted costs from its parent, and possibly adds more – all of which are passed on to its children.