Task Assignment

The **Task Assignment** problem starts with n persons and n tasks, and a known cost for each person/task combination. The goal is to assign each person to an unique task so as to minimize the total cost.

This problem can be solved in polynomial time using an algorithm called the Hungarian Method. However we will develop a Branch and Bound solution as an exploration of some useful B&B techniques.

Decision sequence: At stage i, we will select a task for person i from the tasks not yet assigned
Objective function: As given – minimize the total cost
Initial value of Global Upper Bound: representing the input as a matrix, sum the diagonal elements (i.e. Assign person i to task i, for all i)
Cost-so-far: the sum of the assignments already made
Guaranteed-future-cost: the sum of the least cost possible assignment for each remaining person
Feasible-future-cost: the result of applying the "sum the diagonal" method to the remaining persons and tasks

Consider this very small instance:

	t1	**t2**	**t3**	**t4**	**t5**
p1	4	3	5	4	6
p2	8	4	6	9	3
p3	3	2	4	7	3
p4	7	10	8	8	2
p5	4	8	5	4	3

Initial value of Global Upper Bound U: 4+4+4+8+3 = 23

We can represent each partial solution by the assignments made so far, the cost-so-far, and the rows and columns of the remaining matrix, and the lower and upper bounds.

The first five partial solutions are the result of assigning p1 to each of the 5 available tasks.

Assignments	CSF	Matrix					L	U
1:1	4		**t2**	**t3**	**t4**	**t5**	14	23
		p2	4	6	9	3		
		p3	2	4	7	3		
		p4	10	8	8	2		
		p5	8	5	4	3		

(The yellow highlighted cells are the minimum values in each row; the sum of these is the GFC)

Assignments	CSF	Matrix					L	U
1:2	3		**t1**	**t3**	**t4**	**t5**	14	26
		p2	8	6	9	3		
		p3	3	4	7	3		
		p4	7	8	8	2		
		p5	4	5	4	3		
1:3	5		**t1**	**t2**	**t4**	**t5**	15	26
		p2	8	4	9	3		
		p3	3	2	7	3		
		p4	7	10	8	2		
		p5	4	8	4	3		

1:4	4		**t1**	**t2**	**t3**	**t5**	14	25
		p2	8	4	6	3		
		p3	3	2	4	3		
		p4	7	10	8	2		
		p5	4	8	5	3		

1:5	6		**t1**	**t2**	**t3**	**t4**	23	28
		p2	8	4	6	9		
		p3	3	2	4	7		
		p4	7	10	8	8		
		p5	4	8	5	4		

The algorithm would choose the partial solution with the lowest L value for expansion, then generate new partial solutions by assigning p2 to each of the 4 available tasks. In this example, 1:1,
1:2 and 1:4 are tied for lowest L – we could choose between them randomly, or we could choose the one with the lower U value (in this case, 1:1) in hopes that it will potentially lead to even lower U values, which will let us eliminate other partial solutions.

This algorithm is guaranteed to find the optimal solution (B&B algorithms always do). However we can attempt to improve its efficiency by extracting more information about costs from the matrix.

For example, observe that p1 costs at least 3, no matter which task is assigned. We can subtract 3 from every element of the first row, and add 3 to the CSF of all the initial partial solutions.
Then we can do the same for all the other rows.

We end up with

	t1	**t2**	**t3**	**t4**	**t5**	**Extracted cost**
p1	1	0	2	1	3	3
p2	5	1	3	6	0	3
p3	1	0	2	5	1	2
p4	5	8	6	6	0	2
p5	1	5	2	1	0	3
						Total = 13

So every solution will cost at least 13

But now look at t1, t3 and t4. It will cost at least 1 to assign any worker to t1, at least 2 to assign someone to t3, and at least 1 to assign someone to t4. We can subtract the appropriate value from each of these columns, giving

	t1	**t2**	**t3**	**t4**	**t5**	**Extracted cost**
p1	0	0	0	0	3	3
p2	4	1	1	5	0	3
p3	0	0	0	4	1	2
p4	4	8	4	5	0	2
p5	0	5	0	0	0	3
						Total = 13 `
	1		2	1		**Total = 13 + 1+2+1 = 17**

Thus every solution carries a cost of at least 17.

We can see how this affects the partial solutions developed above:

Assignments	**CSF**	**Matrix**					**L**	**U**
1:1	17		**t2**	**t3**	**t4**	**t5**	17	23
		p2	1	1	5	0		
		p3	0	0	4	1		
		p4	8	4	5	0		
		p5	5	0	0	0		
1:2	17		**t1**	**t3**	**t4**	**t5**	17	26
		p2	4	1	5	0		
		p3	0	0	4	1		
		p4	4	4	5	0		
		p5	0	0	0	0		
1:3	17		**t1**	**t2**	**t4**	**t5**	17	26
		p2	4	1	5	0		
		p3	0	0	4	1		
		p4	4	8	5	0		
		p5	0	5	0	0		
1:4	17		**t1**	**t2**	**t3**	**t5**	17	25
		p2	4	1	1	0		
		p3	0	0	0	1		
		p4	4	8	4	0		
		p5	0	5	0	0		

			t1	t2	t3	t4
1:5	20					
		p2	4	1	1	5
		p3	0	0	0	4
		p4	4	8	4	5
		p5	0	5	0	0

and look at this matrix! It has two rows with all elements > 0, so we can extract even more guaranteed costs: 1 from the first row and 4 from the third row. So our CSF value becomes 25 (previously extracted: 17 + current assignment cost: 3 + new extracted: 5). Thus the partial solution looks like this:

			t1	t2	t3	t4		
1:5	25						25	28
		p2	3	0	0	4		
		p3	0	0	0	4		
		p4	0	4	0	1		
		p5	0	5	0	0		

Since our Global Upper Bound = 23, we can immediately discard this partial solution (and thereby prune off an entire branch of the "tree" of partial solutions).

It may seem that extracting these guaranteed costs is a lot of effort for not much return, but as the example shows, it can help us reduce the set of partial solutions by raising some L values past the Global Upper Bound. In general, the benefit is that this reduces the set of possible solution values for each partial solution – which may eliminate some overlaps. As soon as the L value for some partial solution exceeds the U value for another partial solution, the first one can be discarded because it cannot lead to an optimal solution.

The “guaranteed-cost-extraction” method can help by quickly identifying choices that have high future costs. For example, consider the following very small example:

1	0	1
0	100	50
30	40	0

The selections highlighted in blue and green may look similar, but selecting the blue one creates a reduced matrix where we can extract a cost of 90 (check this) whereas selecting the green one creates a reduced matrix where the extracted cost is only 40 (check this too).

We can apply the same reduction operation each time the new matrix has a row or column where all values are >= 1. This method for computing L takes more time than the simple approach, but hopefully it improves the lower bounds so much that the overall time of the algorithm is reduced.

As mentioned above another way to try to improve the algorithm's efficiency is to use the U values of partial Solutions to break ties, if the partial solutions have equal L values. For example, if two partial solutions had (L,U) pairs (17,25) and (17,23), then we could choose the second one to expand. There is no guarantee that this will lead to the optimal solution faster, but it may. It is important to note that making a deliberate choice in this situation (where the simple algorithm might be considered to be making a random choice between equal options) cannot prevent us from reaching the optimal solution.

Yet another improvement would be to do something more clever than sum the diagonal when computing the feasible-future-cost. For example, a greedy heuristic might be used at this point, such as

for each remaining person, assign the least cost job that has not yet been assigned

(Remember, these are “temporary” assignments for the purpose of establishing U, not “committed” assignments, but they must still be feasible – we can't assign the same job to two persons.)

This might lower the U values for partial solutions, which would give us better information about them, which would in turn allow us to eliminate more partial solutions from consideration, and thereby close in on the optimal solution faster.

0/1 Knapsack – version 1

Item	1	2	3	4	5	6	
Value	130	100	60	60	120	40	
Mass	10	8	5	5	10	5	
							Global U
Value/Mass	13	12.5	12	12	12	8	

Let k = 20

Initial upper bound = greedy solution 110000 Value in = 230 Value out = 280 280

CSF = cost of things we have decided not to take
GFC = cost of things that individually will not fit in the knapsack
FFC = cost of things left on the table after we run a greedy heuristic on the remaining objects

L= CSF + GFC
U = CSF + FFC

In the lines below
(cyan) is used to signify the partial solution with the lowest lower bound
(red) is used to signify a partial solution that we reject as soon as it is generated

		CSF	GFC	L	FFC	U	Global U
PS1	1	0	0	0	280	280	280
PS0	0	130	0	130	160	290	280
							280
At this point PS1 has the lowest lower bound so we choose it for expansion. S now contains:							280
							280
PS0	0	130	0	130	160	290	280

PS11	1	1			0	280	280	280	280	280
PS10	1	0			100	0	100	160	260	260
										260
At this point PS 10 has the lowest lower bound so we choose it for expansion. S now contains:										260
										260
PS0	0				130	0	130	160	290	260
PS11	1	1			0	280	280	280	280	260
PS101	1	0	1		100	120	220	160	260	260
PS100	1	0	0		160	0	160	120	280	260
										260
At this point PS0 has the lowest lower bound so we choose it for expansion. S now contains:										260
										260
PS11	1	1			0	280	280	280	280	260
PS101	1	0	1		100	120	220	160	260	260
PS100	1	0	0		160	0	160	120	280	260
PS01	0	1			130	0	130	160	290	260
PS00	0	0			230	0	230	40	270	260
										260
At this point PS01 has the lowest lower bound so we choose it for expansion. S now contains:										260
										260
PS11	1	1			0	280	280	280	280	260
PS101	1	0	1		100	120	220	160	260	260
PS100	1	0	0		160	0	160	120	280	260
PS00	0	0			230	0	230	40	270	260
PS011	0	1	1		130	120	250	160	290	260
PS010	0	1	0		190	0	190	120	310	260
										260
At this point PS100 has the lowest lower bound so we choose it for expansion. S now contains:										260
										260
PS11	1	1			0	280	280	280	280	260
PS101	1	0	1		100	120	220	160	260	260
PS00	0	0			230	0	230	40	270	260
PS011	0	1	1		130	120	250	160	290	260
PS010	0	1	0		190	0	190	120	310	260
PS1001	1	0	0	1	160	120	280			260

PS1000	1	0	0	0		220	0	220	40	260	260
											260
At this point PS010 has the lowest lower bound so we choose it for expansion. S now contains:											260
											260
PS11	1	1				0	280	280	280	280	260
PS101	1	0	1			100	120	220	160	260	260
PS00	0	0				230	0	230	40	270	260
PS011	0	1	1			130	120	250	160	290	260
PS1000	1	0	0	0		220	0	220	40	260	260
PS0101	0	1	0	1		190	120	310			260
PS0100	0	1	0	0		250	0	250	40	290	260
											260
At this point PS101 has the lowest lower bound so we choose it for expansion. S now contains:											260
											260
PS11	1	1				0	280	280	280	280	260
PS00	0	0				230	0	230	40	270	260
PS011	0	1	1			130	120	250	160	290	260
PS1000	1	0	0	0		220	0	220	40	260	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS1010	1	0	1	0		160	120	280			260

PS1000 has the lowest lower bound.

PS11	1	1				0	280	280	280	280	260
PS00	0	0				230	0	230	40	270	260
PS011	0	1	1			130	120	250	160	290	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS10001	1	0	0	0	1	220	40	260	40	260	260
PS10000	1	0	0	0	0	340	0	340			260
PS00 has the lowest lower bound.											260
											260
PS11	1	1				0	280	280	280	280	260

PS011	0	1	1			130	120	250	160	290	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS10001	1	0	0	0	1	220	40	260	40	260	260
PS001	0	0	1			230	0	230	40	270	260
PS000	0	0	0			290	0	290			260
											260
PS001 has the lowest lower bound.											260
											260
PS11	1	1				0	280	280	280	280	260
PS011	0	1	1			130	120	250	160	290	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS10001	1	0	0	0	1	220	40	260	40	260	260
PS0011	0	0	1	1		230	0	230	40	270	260
PS0010	0	0	1	0		290	0	290			260
											260
PS0011 has the lowest lower bound											260
											260
PS11	1	1				0	280	280	280	280	260
PS011	0	1	1			130	120	250	160	290	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS10001	1	0	0	0	1	220	40	260	40	260	260
PS00111	0	0	1	1	1	230	40	270			
PS00110	0	0	1	1	0	350	0	350			
											260
PS011 has the lowest lower bound											260
											260
PS11	1	1				0	280	280	280	280	260
PS0100	0	1	0	0		250	0	250	40	290	260
PS1011	1	0	1	1		100	160	260	160	260	260
PS10001	1	0	0	0	1	220	40	260	40	260	260
PS0111	0	1	1	1		130	160	290			
PS0110	0	1	1	0		190	120	310			

												260
PS0100 has the lowest lower bound												260
												260
PS11	1	1					0	280	280	280	280	260
PS1011	1	0	1	1			100	160	260	160	260	260
PS10001	1	0	0	0	1		220	40	260	40	260	260
PS01001	0	1	0	0	1		250	40	290		250	260
PS01000	0	1	0	0	0		370	0	370		370	260
												260
We expand PS1011												260
												260
PS11	1	1					0	280	280	280	280	260
PS10001	1	0	0	0	1		220	40	260	40	260	260
PS10111	1	0	1	1	1		not feasible					260
PS10110	1	0	1	1	0		220	40	260	40	260	260
												260
We expand PS10001												260
												260
PS11	1	1					0	280	280	280	280	260
PS10110	1	0	1	1	0		220	40	260	40	260	260
SP100011	1	0	0	0	1	1	not feasible					260
PS100010	1	0	0	0	1	0	260	0	260	0	260	260
									0		0	260
We expand PS10110												260
												260
PS11	1	1					0	280	280	280	280	260
PS100010	1	0	0	0	1	0	260	0	260	0	260	260
PS101101	1	0	1	1	0	1	not feasible					260
PS101100	1	0	1	1	0	0	260	0	260	0	260	260

We expand PS100010 – but wait! It is a complete solution – so we know it the optimal solution. We are done.

Total number of partial solutions generated: 34

Total number of potential complete solutions: 64

Branch and bound reduced the total amount of work by about 50%

Note regarding PS11:

When the Global Upper bound U drops to 260, PS11 *could* be pruned. However finding all such redundant partial solutions would require searching and updating the data structure being used to hold S. For large-scale problems this might be worth doing since we always want to keep S small if possible. For smaller problems the overhead involved in always keeping S small may not be worth it.

0/1 Knapsack – version 2

In Version 1 of our solution to this problem we used a very simple method of computing GFC, and we did not use any particular method of breaking ties between partial solutions with the same L value

In Version 2 we will improve GFC by first eliminating any objects that won't fit, then putting the remaining objects in groups that cannot all go in
– we know we must leave out at least one object from each group
We will also break ties in favour of the partial solution that is closest to being a complete solution
– ie in case of a tie, we will choose the partial solution that includes more decisions

Item	1	2	3	4	5	6
Value	130	100	60	60	120	40
Mass	10	8	5	5	10	5
Value/Mass	13	12.5	12	12	12	8

Global U

Let k = 20

Initial upper bound =greedy solution 110000 Value in 230 Value out = 280 280

CSF = cost of things we have decided not to take
GFC = as defined above
FFC = cost of things left on the table after we run a greedy heuristic on the remaining objects

L= CSF + GFC
U = CSF + FFC

In the lines below
is used to signify the partial solution with the lowest lower bound
is used to signify a partial solution that we reject as soon as it is generated

				CSF	GFC	L	FFC	U	
PS1	1			0	120	120	280	280	280
PS0	0			130	60	190	160	290	

At this point PS1 has the lowest lower bound so we choose it for expansion. S now contains:

PS0	0			130	60	190	160	290	
PS11	1	1		0	280	280	280	280	
PS10	1	0		100	60	160	160	260	260

At this point PS 10 has the lowest lower bound so we choose it for expansion. S now contains:

PS0	0			130	60	190	160	290	
PS11	1	1		0	280	280	280	280	see note below
PS101	1	0	1	100	160	260	160	260	
PS100	1	0	0	160	60	220	120	280	

At this point PS0 has the lowest lower bound so we choose it for expansion. S now contains:

PS11	1	1		0	280	280	280	280
PS101	1	0	1	100	160	260	160	260
PS100	1	0	0	160	60	220	120	280
PS01	0	1		130	60	190	160	290
PS00	0	0		230	40	270		

At this point PS01 has the lowest lower bound so we choose it for expansion. S now contains:

PS11	1	1		0	280	280	280	280
PS101	1	0	1	100	160	260	160	260
PS100	1	0	0	160	60	220	120	280
PS011	0	1	1	130	160	290		
PS010	0	1	0	190	60	250	120	310

At this point PS100 has the lowest lower bound so we choose it for expansion. S now contains:

PS11	1	1					0	280	280	280	280
PS101	1	0	1				100	160	260	160	260
PS010	0	1	0				190	60	250	120	310
PS1001	1	0	0	1			160	120	280		
PS1000	1	0	0	0			220	40	260	40	260

At this point PS010 has the lowest lower bound so we choose it for expansion. S now contains:

PS11	1	1					0	280	280	280	280
PS101	1	0	1				100	160	260	160	260
PS1000	1	0	0	0			220	40	260	40	260
PS0101	0	1	0	1			190	120	310		
PS0100	0	1	0	0			250	40	290		

At this point PS101 and PS1000 are tied. We choose PS1000 because it contains more decisions.

PS11	1	1					0	280	280	280	280
PS101	1	0	1				100	160	260	160	260
PS10001	1	0	0	0	1		220	40	260	40	260
PS10000	1	0	0	0	0		340	0	340		

PS101 and PS10001 are tied. We choose PS10001 because it contains more decisions.

PS11	1	1					0	280	280	280	280
PS101	1	0	1				100	160	260	160	260
PS100011	1	0	0	0	1	1	infeasible				260
PS100010	1	0	0	0	1	0	260	0	260	0	260

PS1011 and PS100010 are tied. We choose PS100010 because it contains more decisions.

Oh! PS100010 is a complete solution, and we have selected it as having the lowest L value in S.   We know it is an optimal solution.

Total number of partial solutions generated: 18

Total number of potential complete solutions: 64

Branch and bound reduced the total amount of work to less than 30% of the work involved in exhaustive search of the full set of potential solutions.

This example shows that with a little more work in computing L and U for each partial solution, we can greatly accelerate the search for an optimal solution.

Note regarding PS11:
When the Global Upper bound U drops to 260, PS11 *could* be pruned. However finding all such redundant partial solutions would require searching and updating the data structure being used to hold S. For large-scale problems this might be worth doing since we always want to keep S small if possible. For smaller problems the overhead involved in always keeping S small may not be worth it.