Tutorial Sections 3-4

1. Imagine you were awarded the opportunity to make a tour across the UK next vacation. Starting from Birmingham, you plan to explore Edinburgh, Cardiff, and Newcastle. However, with time as a valuable commodity, you aim to minimize the total travel distance, ensuring you have more time to enjoy each destination. The approximate distances between the cities are shown on the map of Figure 1, in miles.
   (a) Perform an exhaustive search to find the shortest route that visits each city once and returns to the starting point, Birmingham. Construct the computational graph illustrating the exhaustive SDP, highlighting the optimal route with its corresponding distance.
   (b) Implement a greedy search strategy to solve the same problem. Is the result exact? Construct the computational graph for this greedy approach.
   (c) Identify and explain the complexity classes of the algorithm in (a) and (b) above. How does the complexity class affect the performance and efficacy of each approach and their results?

Figure 1: Problem 2. The numbers on the map represent the distance between the cities.

Solution: This is another example of enumerating permutations in a contextualised version of the travel salesman problem.

(a) Again, we can follow the steps from the SDP algorithm (4.1) from the notes:

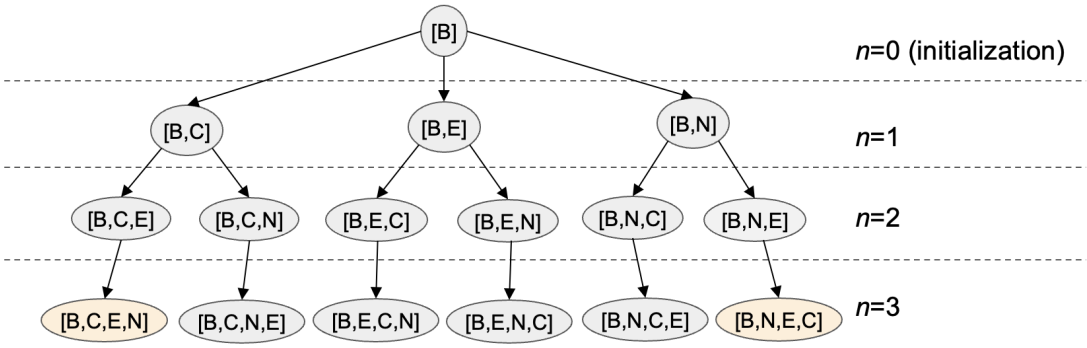
1. **Initialization**: at $n = 0$, we have an empty path starting from Birmingham (B); so our root configuration in the set of candidate configurations is $S = \{[B]\}$;
2. **Extension**: we can extend each configuration in S by adding one of the remaining cities in each iteration;
3. **Reduction**: for this specific problem, every path can be extended to an optimal configuration, so we won't actively reduce configurations.
4. **Iteration**: we will repeat the extension step until all cities have been visited;

5. **Select Best**: we will choose the configuration with the shortest total distance.

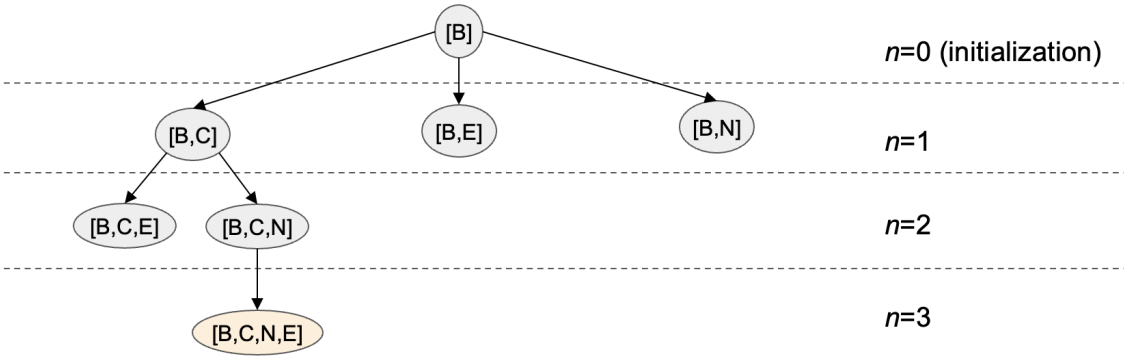
For $n = 1$, we have $S = \{[B, C], [B, E], [B, N]\}$;
For $n = 2$, we have $S = \{[B, C, E], [B, C, N], [B, E, C], [B, E, N], [B, N, C], [B, N, E]\}$;
For $n = 3$, we have
$S = \{[B, C, E, N], [B, C, N, E], [B, E, C, N], [B, E, N, C], [B, N, C, E], [B, N, E, C]\}$;
Given the distances in the map, we can compute the total distance for each configuration:

- $[B, C, E, N] = 100 + 400 + 120 + 200 = 820$ miles
- $[B, C, N, E] = 100 + 320 + 120 + 300 = 840$ miles
- $[B, E, C, N] = 300 + 400 + 320 + 200 = 1220$ miles
- $[B, E, N, C] = 300 + 120 + 320 + 100 = 840$ miles
- $[B, N, C, E] = 200 + 320 + 400 + 300 = 1220$ miles
- $[B, N, E, C] = 200 + 120 + 400 + 100 = 820$ miles

Therefore, the *Select Best* step would pick the optimal solution as either B $\rightarrow$ C $\rightarrow$ E $\rightarrow$ N $\rightarrow$ B or B $\rightarrow$ N $\rightarrow$ E $\rightarrow$ C $\rightarrow$ B. The computational graph for this solution is presented below:



(b) In the greedy search approach, we choose the option that is the currently best, without considering the global optimum (which translates to always travelling next to the nearest unvisited city from our current location). This strategy modifies the *Step 3. Reduction* of the algorithm above; now we choose the closest city that hasn't been visited yet.
At $n = 0$, we keep the same configuration: $S = \{[B]\}$;
At $n = 1$, we still have $S = \{[B, C], [B, E], [B, N]\}$. The closest city among these options is Cardiff (100 miles). Therefore, the reduction step keeps the configuration $[B, C]$ and removes all others;
At $n = 2$, the possibilities are $S = \{[B, C, E], [B, C, N]\}$. The reduction step keeps the configuration $[B, C, N]$;
At $n = 3$, the only possibility is $[B, C, N, E]$. Therefore, the optimal solution yields a different route: B $\rightarrow$ C $\rightarrow$ N $\rightarrow$ E $\rightarrow$ B, with a total of 840 miles.
This solution is not exact, as can be seen by comparison with exhaustive search. As discussed in Section 4, this greedy approach finds a suboptimal solution, which means that the modified reduction step is not valid, so it is not guaranteed to find the globally optimal solution.
The computational graph for this approach is shown below:



(c) The exhaustive search generates all possible permutations of the cities and then selects the one with the shortest total distance. The time complexity is primarily determined by the number of permutations generated. For n cities, there are $n!$ permutations. Therefore, the time complexity is $O(n!)$. Similarly, the space complexity is also $O(n!)$, as it needs to store all possible permutations.
The greedy search algorithm makes the locally optimal choice at each stage, choosing the closest next city. At each step, the algorithm chooses from among the remaining cities. This requires comparing distances to $n-1$ cities in the first step, $n-2$ in the second, and so on. The time complexity is thus $O(n^2)$ in the worst case. The space complexity is lower, as it only needs to store the current best path at any time. This is $O(n)$.
Although the greedy algorithm does not guarantee the shortest possible route, it is more computationally efficient than the exhaustive search (this is more relevant as the number of cities increases).

2. You are planning a shopping trip on foot to a store where everything is on sale, which means you are willing to get the best value for your money within a limited carrying capacity. The shop stocks four items from your shopping list:

Item	A	B	C	D
Weight (kg)	7	3	4	5
Price (£)	42	12	40	25

The challenge is to select items that maximise the total value without exceeding the 10 kg weight capacity of your shopping bag, keeping in mind that you only need one of each item. Your tasks are as follows:

(a) Employ an exhaustive search strategy to enumerate the configuration set of possible solutions for the problem and identify the combination of items that maximises the total price without exceeding the weight limit of your bag.

(b) Identify and discuss the complexity class of the exhaustive search in this problem.

(c) Another method to evaluate the items is by their value-to-weight ratio. Using this criterion, apply a greedy SDP to solve the same problem. Does the answer differ from the exhaustive search outcome?

(d) For this specific scenario, the optimal solution using the greedy approach coincidentally finds the globally optimal solution. But it cannot be the case that this greedy SDP is always globally exact. Explain why this cannot be generally exact, and provide an example scenario where the greedy SDP would not be exact.

Solution: This is a simplified and contextualised version of the *knapsack* problem in which, coincidentally, the greedy search finds a globally optimal solution. Here, the aim is to maximise the total value (price) of items with a constraint (i.e., without exceeding a weight limit).

(a) We can generate all possible combinations of items A, B, C, and D and calculate the total weight and price for each combination. We can easily verify that the possible solutions whose total weight is ≤ 10 are:

$$\mathcal{X} = \{\{A(7kg, £42)\}, \{B(3kg, £12)\}, \{C(4kg, £40)\}, \{D(5kg, £25)\}, \{A, B(10kg, £52)\}, \{B, C(7kg, £52)\}, \{B, D(8kg, £37)\}, \{C, D(9kg, £65)\}\}$$

The solutions

$$\mathcal{X}' = \{\{A, C(11kg)\}, \{A, D(12kg)\}, \{A, B, C(14kg)\}, \{A, B, D(15kg)\}, \{A, C, D(16kg)\}, \{B, C, D(12kg)\}, \{A, B, C, D(19kg)\}\}$$

are not feasible with the given weight constraint. The optimal solution from $\mathcal{X}$ is $\{C, D\}$, which weighs 9 kg and costs £ 65.
Based on Algorithm 4.1 from the notes for an exhaustive search strategy, we can find the optimal solution with the following steps:

1. **Initialization**: start with an empty bag ($n = 0$);
2. **Extension**: we will set $n = n + 1$ and use data item x_n to extend all candidate configurations in S;
3. **Reduction**: remove configurations that exceed the weight limit;
4. **Iteration**: repeat until all items have been considered
5. **Select best**: choose the combination with the highest total value that doesn't exceed the weight limit.

At $n = 0$, we have $S = \emptyset$;
At $n = 1$, we have $S = \{\{A\}, \{\}\}$, for a total of 7 kg and £ 42, which won't be removed;
At $n = 2$, $S = \{\{A, B\}, \{A\}, \{B\}, \{\}\}$, and no candidate configuration will be removed at this step;
At $n = 3$, we have $S = \{\{A, B, C\}, \{A, B\}, \{A, C\}, \{A\}, \{B, C\}, \{B\}, \{C\}, \{\}\}$ after extension; after reduction, $S = \{\{A, B\}, \{A\}, \{B, C\}, \{B\}, \{C\}, \{\}\}$, as other configurations exceed the weight limit;
At $n = 4$,
$S = \{\{A, B, D\}, \{A, B\}, \{A, D\}, \{A\}, \{B, C, D\}, \{B, C\}, \{B, D\}, \{B\}, \{C, D\}, \{C\}, \{D\}, \{\}\}$; after reduction, $S = \{\{A, B\}, \{A\}, \{B, C\}, \{B, D\}, \{B\}, \{C, D\}, \{C\}, \{D\}, \{\}\}$
The *select best* step computes the total value for each configuration, which results in:

- $\{A, B\} = 10$ kg, £ 52;
- $\{A\} = 7$ kg, £ 42;
- $\{B, C\} = 7$ kg, £ 52;
- $\{B, D\} = 8$ kg, £ 37;
- $\{B\} = 3$ kg, £ 12;
- $\{C, D\} = 9$ kg, £ 65;
- $\{C\} = 4$ kg, £ 40;
- $\{D\} = 5$ kg, £ 25;

Therefore, the optimal configuration is $X^* = \{C, D\}$.

(b) In the exhaustive search approach above, each item can either be included or excluded from the shopping bag, which creates a binary decision tree. For N items, there will be 2^N possible combinations (since each item has 2 "states": included or excluded). For each combination, the algorithm calculates the total weight and total price, which also takes $O(N)$ time in the worst case. The space complexity is mainly due to storing all 2^N combinations. Therefore, the exhaustive search algorithm for this problem is exponential in both time and space complexity. As the number of items increases, the resources required to compute and store all possible combinations increase exponentially, resulting in an intractable problem.

(c) Using a greedy search and considering our objective to maximise the value-to-weight ratio of the items, we can start with an empty configuration and add the item with the highest value-to-weight ratio that hasn't been included yet (*extension*) and doesn't exceed the weight limit (*reduction*). We continue until all items are considered (*iteration*), and the resulting configuration will be the outcome of the greedy strategy.
The value-to-weight ratio of each item is: A (£ 6/kg), B (£ 4/kg), C (£ 10/kg), D (£ 5/kg).
At $n = 0$, we have $S = \{\}$. At $n = 1$, we pick C as this item has the highest value-to-weight ratio among all items. This adds adds 4 kg in our bag and is worth £ 40.
At $n = 2$, we would pick A, but this candidate configuration will be removed as it exceeds the weight limit.
At $n = 3$, we the configuration $\{C, D\}$ is a valid choice with a total of 9 kg and £ 65.
At $n = 4$, there would be B to pick but it will be removed due to the weight constraint.
Therefore, the optimal solution using the greedy approach coincidentally finds the globally optimal solution.

(d) The greedy strategy is not always globally exact due to its nature of making local optimum choices at each step without considering the overall or future implications of these choices. Once a choice is made in a greedy SDP, it does not reconsider this decision. However, these local optima do not necessarily lead to a global optimum. A choice that seems best in the short term may exclude better options later on.
We can see this in practice with a simple variation of the items. For example, suppose we had:

- Item X: 9 kg, £ 80 (value-to-weight ratio = 8.89);
- Item Y: 6 kg, £ 50 (value-to-weight ratio = 8.33);
- Item Z: 4 kg, £ 35 (value-to-weight ratio = 8.75);

The greedy algorithm would pick item X first due to its highest ratio, leaving only 1 kg of capacity, which is insufficient to add any other item. However, by considering all configurations, $\{Y, Z\}$ would yield a higher total value (£ 85) within the same weight limit.