20191112

# One Last B&B Example – A Practical Application

Now that we have seen the "matrix reduction" idea, let's consider a really difficult problem: the Traveling Salesperson Problem, or TSP. In this problem we imagine a salesperson who has to choose a route that takes them around a cycle of n cities, visiting each city once and returning to where they started. All the cities are connected by direct roads and all the roads have different costs. The goal is to find the route with the smallest total cost. A small example might look like this. The integers represent the cost of traveling from the "row" city to the "column" city:

| | Burgerburgh | Pizzapolis | Nachoville | Timbitton |
|---|---|---|---|---|
| Burgerburgh | $\infty$ | 15 | 13 | 8 |
| Pizzapolis | 15 | $\infty$ | 4 | 11 |
| Nachoville | 13 | 4 | $\infty$ | 14 |
| Timbitton | 8 | 16 | 14 | $\infty$ |

Note the $\infty$ entries on the diagonal – this is just a simple way of making sure we never try to go from a city to itself. Basically any solution that tries to do this ends up with an infinitely large cost so it is rejected immediately – we don't need to code anything extra to handle this.

We can think of the row for Burgerburgh as representing the different costs of going from Burgerburgh to any of the other cities … and we can reduce the row by subtracting off the smallest of them. Of course we do the same for all the rows. Similarly we can think of the columns as representing the costs to arrive at the cities. If any column is all > 0 after the rows are reduced we can reduce that column as well.

The matrix reduces to :

| | Burgerburgh | Pizzapolis | Nachoville | Timbitton |
|---|---|---|---|---|
| Burgerburgh | $\infty$ | 7 | 5 | 0 |
| Pizzapolis | 11 | $\infty$ | 0 | 7 |
| Nachoville | 9 | 0 | $\infty$ | 10 |
| Timbitton | 0 | 8 | 6 | $\infty$ |

This gives us an extracted cost of 8+4+4+8 = 24

For our initial value of $U_{Global}$ we can use a Greedy Heuristic as we did before, but note that the solution must be a tour of the cities. Starting at Burgerburgh we can choose the 0-cost road to Timbitton, but we cannot follow that with the 0-cost road from Timbitton back to Burgerburgh because that takes us home too soon – we have to visit all the cities before we return. So our Greedy choice now is the 6-cost road from Timbitton to Nachoville. This has to be followed by the 0-cost road from Nachoville to Pizzapolis and finally the 11-cost road from Pizzapolis to Burgerburgh. Our initial $U_{Global}$ value is 24+17 = 41 (the 24 comes from the extracted cost).

We'll just explore the progress of the algorithm a bit. Our first decision is where to go from Burgerburgh. The first option is Pizzapolis, which costs 7 so our CSF is 24+7=31. When we reduce the matrix to represent this selection, we

- remove the row for Burgerburgh (we have decided how to leave this city)
- remove the column for Pizzapolis (we have decided how to arrive at this city)
- set the entry for Pizzapolis $\rightarrow$ Burgerburgh to $\infty$ because we ***cannot*** use this edge – it would take us home too soon

This gives us

| | Burgerburgh | Nachoville | Timbitton |
|---|---|---|---|
| Pizzapolis | $\infty$ | 0 | 7 |
| Nachoville | 9 | $\infty$ | 10 |
| Timbitton | 0 | 6 | $\infty$ |

And look! The row for Nachoville has all values $\geq 9$ so we can extract a cost of 9 – after which the column for Timbitton has all values $\geq 1$ so we can extract a cost of 1. We can add

this total extracted cost of 10 to our $l_P$ value for this partial solution. We can use the Greedy Heuristic again to get a value for $u_P$.

As we progress through the algorithm we lose rows and columns of the matrix, we set some values in the table to $\infty$ and we reduce others to 0. Eventually we can reach a point in each partial solution where all the remaining costs are either $\infty$ or 0. At this point the partial solution becomes a full solution by choosing any feasible extension – there are no further costs. Such a full solution is either discarded because its cost is too high, or becomes the current best solution.

The algorithm is guaranteed to find an optimal solution. If we have done a good job of defining the lower and upper bounds, it will often find an optimal solution relatively quickly.

This is important because we absolutely do not have a polynomial-time algorithm for the TSP problem. There are some restricted cases where we can find the optimal solution easily, and some cases where we can get a good approximation to the optimal answer quite quickly, but the general TSP problem is one of the hardest problems we know of – which means that nobody realistically hopes that we will ever find a fast algorithm that solves it (at least not until quantum computing becomes a reality). The best algorithm to find the optimal solution to a general case TSP problem is – you guessed it – Branch and Bound.