#### Traveling Salesman Problem

The traveling salesman problem has long been studied as an intro to algorithms and big-O notation. It is a problem that many people, even those not studying computer science, may have heard about. This problem involves a traveling salesperson that needs to visit n cities to sell their merchandise. To optimize his time, and therefore his profits, he needs to plan out the route that is the shortest possible route between the cities. Traditional forms of the problems limit the salesperson to a single visit to each city, whereas other forms allow the salesperson to pass through a city multiple times if it yields a faster route.

This is a very rigorously studied problem as it fully demonstrates the property of NP-complete because all possible combination of routes must be looked at. Not only that, but this problem affects billions of people every single day. Without optimal path planning, Amazon’s costs could go up, which in turn causes their merchandise to increase in price which affects all their users.

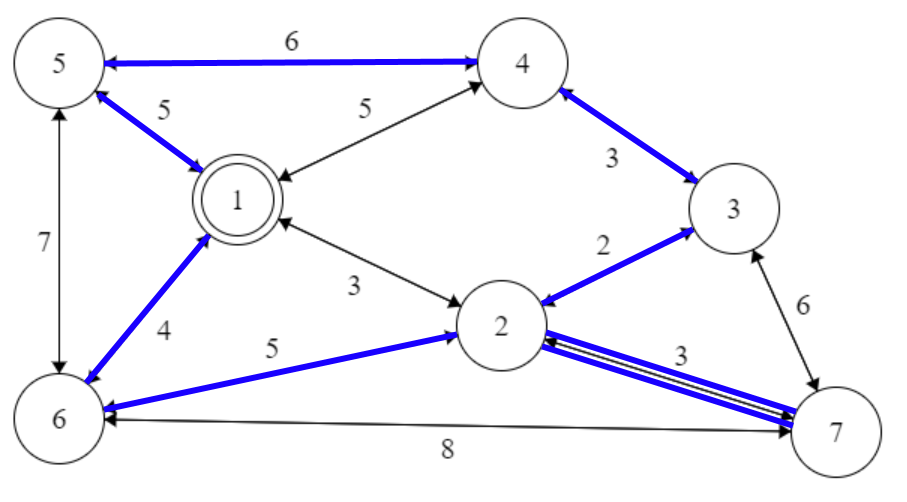
A brute force search for the optimal route would result in an algorithm that has a running time of O(n!) where n is the number of cities that need to be visited. This makes a brute force algorithm non-feasible even when the number of cities is only 15. Linear programming techniques work well for up to 200 cities, but the current method for solving large instances is an approach using a derivative of a branch-and-bound algorithm called a branch-and-cut algorithm. This solution holds the current record, solving an instance with 85,900 unique cities[[1]](#footnote-1)­.

Figure 2. An example of a solved traveling salesman problem starting at city 1 and ending back at city 1. This example demonstrations the version of the problem where the salesman can backtrack if doing so would result in a more optimal path.

#### Job Shop Scheduling Problem

The job shop scheduling problem is another very popular problem that is shared with students during their architecture and operating systems classes because it is an impressive optimization problem that severely impacts computers today.

The problem focuses on scheduling computer tasks (jobs) in such a way that optimizes performance. What makes the job shop scheduling problem so tricky is that the method of optimizing performance is different for many different systems based on the objective of that machine. For example, if we have a uniprocessor system we need to determine if multiprogramming is important to us, or if we want to frontload all processing power to singular tasks.

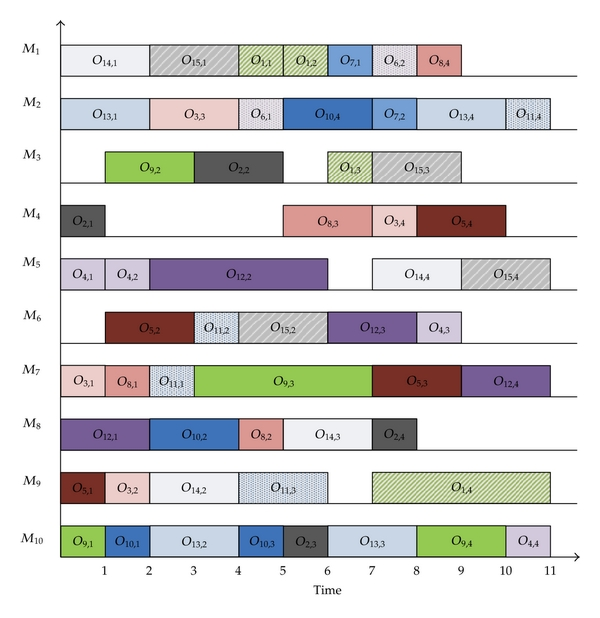
The job shop scheduling problem encapsulates the traveling salesman problem. The traveling salesman problem is a special case of a job shop scheduling problem where the salesman is a machine, and the cities are the jobs. Knowing that the traveling salesman problem is NP-Hard, we can infer that the job shop scheduling problem is also NP-Hard.

Figure 3. This is an example of a job shop scheduling problem with each job is a specific color and each objective for each job must be completed in order (denoted by Oj,n where j is the job and n is the order at which it must complete. This example uses 10 machines to optimize the schedule.

The problem consists of n jobs J­1, J2, …, Jn each with a set of operations O1, O2, …, On that need to be processed in a specific order. Throughout the variations of this problem the many constraints include, but are not limited to: each operation having a specific machine that it needs to be processed on, is multiprocessing available, focus on minimizing the average response time, job dependencies, deterministic or probabilistic processing times, and minimizing the total length of the schedule.

## EVALUATION METRICS

Due to the nature of these problems, it is important to evaluate multiple dimensions of complexity to fully understand the implications of these problems in large scale. Therefore, we have recorded and studied the three dimensions of time, space, and accuracy. By considering multiple dimensions, we can discover relationships and correlations that will further our research.

### Time

Time is a tricky evaluation metric due to factors such as computer specifications, project implementation, and outside interference. Therefore, we are logging both real-time as well as any data pertaining to the algorithm. Specifically: array accesses, array copies, swaps, failed branches, etc. By recording this additional information, an understanding can be made of the relationships between different heuristics for each problem.

### Space

Easier than time, space can be calculated by determining the total memory usage used to calculate the problem. However, evaluating space needs to consider maximum space required during the algorithm and average space required during the algorithm to encompass the problem solution more fully. To do this, data structure sizes and count of data structures in use will be calculated to provide a reasonable space evaluation.

### Accuracy

These problems indeed have optimal solutions, but to find the perfect answer is exponentially more complex and time consuming than using heuristics for educated estimates. Therefore, our heuristics might not be 100% correct 100% of the time. For many of these problems, estimates are more than enough to claim that the problem is satisfied, but at which point is an estimate not good enough? Specifically, at what point of accuracy is that estimate not valid? To record and evaluate accuracy, we will simply be comparing to the theoretical optimal solution for each problem. Therefore, Accuracy = Our Solution / Theoretical Optimal Solution.

1. Applegate et al. (2006) [↑](#footnote-ref-1)