TRAVELING SALESMAN NP-HARD PROBLEM AND USING OR TOOL THAT SOLVES INSTANCES USING OFF-THE-SHELF SAT SOLVERS

FINAL PROJECT : THEORY OF COMPUTATION (CS_517_001_S2022)

Name: Rajapuram Nithish Kumar Reddy **Osu Id: 934369743**

Name: Preethi Kurapaty **Osu Id : 934371390**

INTRODUCTION

The worldwide ecosystem is in continuous progression and technological advancement has reached the breakthrough phase. The field of computer science holds the momentous role to make every aspect possible and feasible. The theoretical background, the designing, development as well as implementation of the software programming empowers the entire ecosystem. Various programming languages have been introduced which has brought prodigious walkover in every field. Computer science is trending in various specializations including machine learning and Artificial Intelligence, Cybersecurity, Data Science, Business Information Technology, Healthcare Management, Digital Content Administration, Library and Information Science along with Software Engineering. These computational systems power the whole world in this generation.

Computer programming methods and mathematical optimization provide the ultimate solution for complex problems in the field of computer science. It delivers the ability to decide and manipulate certain intricate situations which is not possible to detect through manual interventions. Nevertheless, computer programming does that in a few minutes or a fraction of seconds like dynamic programming. Apart from that, there is a number of unified toolkits present considering SAT solvers which have the ability to solve complex NP-hard problems. This academic article is based on the traveling salesman's NP-hard problem and building a tool that solves instances using off-the-shelf SAT solvers. The shortest route is determined through which the salesman will travel to move from one location to another. All the located points will be covered distance-wise without any repetition. The python-based SAT solver approach using OR tool is used to solve the non-deterministic polynomial-time hard problem (Dahiya et al., 2018).

Dynamic Programming

Dynamic programming is a significant computer programming approach as well as a mathematical optimization method which to simplify any complex problems through breaking them into meeker sub-problems in a recursive way. The actual idea is to store the outcomes of the sub-problems so that during the forthcoming requirements, there will be no need of recomputing them. This method of loading the subproblems and the value is known as memoization. Through the procedure of saving the different values in the array, the time for calculations of sub-problems is saved which have previously come transversely. The process of optimization deduces time intricacies from exponential to polynomial. Intending towards the illustration, the traveling salesman problem is optimized through a dynamic programming algorithm to find the shortest pathway. Dynamic programming through memoization is fundamentally a top-down method of dynamic programming. By reversing the course, in the way the algorithm works, that is commencing from the base instance whereas working on the way to the resolution, dynamic programming can be implemented in a bottom-up manner. The most prominent benefit of using dynamic programming is that it delivers effectual algorithms for NP-hard problems like traveling sales problems (Kool et al., 2021).

```
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}
```

```
f[0] = 0;
f[1] = 1;

for (i = 2; i <= n; i++)
{
    f[i] = f[i-1] + f[i-2];
}

return f[n];
```

Figure: Comparison of Exponential Recursion and Linear Dynamic Programming (Kool et al., 2021).

NP-hard Problem

The theory of computational complexity deals with the different classification of problems according to the use of resources while relating these classifications with each other. Considering this theory, the non-deterministic polynomial-time hardness is denoted to the property of any class of complex problems which are considered to be the extreme hard problems as hard as the problems in the non-deterministic polynomial time. The most common problem is the traveling salesman problem. A decision problem (Yes or No) H is considered to be NP-hard when for each problem L in NP, a polynomial-time many-one reduction is present from L to H. To be specific, each problem L in NP can be resolved in polynomial time through a prophecy machine by a revelation for H. The problems belong to the class of complex problems that are at least as hard as the toughest complications in non-deterministic polynomial time. Problems that are NP-hard do not partake to be the fundamentals of NP; certainly, they might not even be decidable. Apart from NP-hard, there are many more problem categories including NP, NP-complete, NP-easy, NP-equivalent as well as NP-intermediate problems. The Boolean satisfiability problem (SAT) solver is frequently and mostly used to solve the non-deterministic polynomial-time hard problems (Kim et al., 2021).

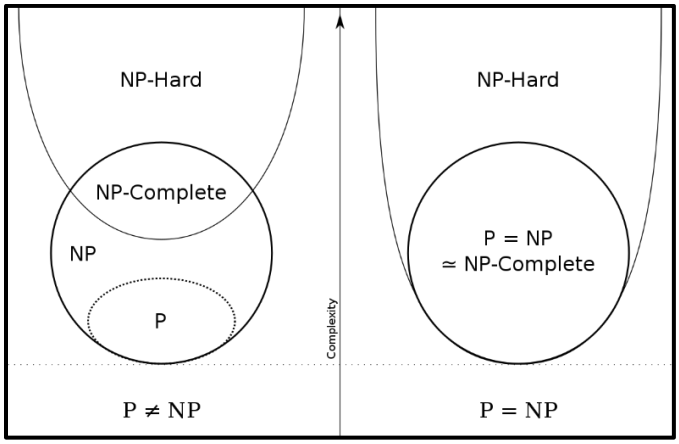

Figure: NP-hardness (Kim et al., 2021).

OR Tools

The google OR-tools is mainly a software program that is open-source and used in optimization and tackling the hardest problems considering routing of vehicles and persons, the flows, linear programming, and integers as well as restraint programming. The OR tools are used in this program to solve the traveling salesman problem. The data set of rules and the library includes extensive diversity of heuristics aimed at diverse problems primarily grounded on the exceptional restraint programming solver of Google. The dataset might be used for an extensive variety of problems. The instances of the traveling salesman problem are solved through the provided tsblib library files making use of the heuristics delivered by the OR-tools of Google. A python integrated running environment is used to solve the instances. Aiming towards the prime technique, the traveling salesman problem instance is overloaded, in addition, the distance matrix is formed. At that moment, the problem is tendered over to the OR-tools direction-finding solver as a Routing Archetypal, composed through the default search constraint set (Chen et al., 2019).

SAT Solver

An SAT solver is considered to be an algorithm for executing satisfiability. It uses the Boolean logic formulation as the input besides returning SAT when it invents an amalgamation of data variables that has the ability to gratify it otherwise UNSAT, when it can prove that there is no such amalgamation occurs. The Boolean satisfiable problem query about at least a sole combination of input binary variables that is $x_i \in$ {false, true. Furthermore, towards solving a problem related to SAT, the Boolean logic formulation is converted to a standard form which is considered to be more agreeable towards algorithmic operations (Selsam et al., 2018).

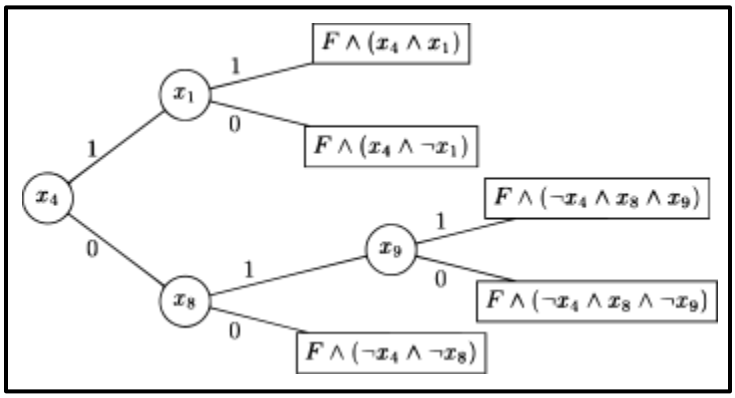

Figure: SAT Solver

TRAVELING SALESMAN PROBLEM

The traveling salesman problem is considered to be a noteworthy and standard problem in view of combinative optimization. This problem is based on the routing system and to determine the shortest way that a salesman will take to travel to different cities starting from the origin and returning to the origin city. The distance between the cities is delivered. This problem is an NP-hard problem and considering the real-life scenario, there is a need for sub-optimal solutions instead of optimal solutions. The individuals who are directly related to the logistics and planning sector, really care about this problem to be solved in a real-life scenario. Aiming towards a real-life illustration, a program manager of any band manages and schedule a number of the program in various cities with a full-proof plan to avoid unnecessary time consumption as well as cost. Owing to this reason, they are keen to find the shortest path to travel the least distance and help the entire crew not to get exhausted at the end of the day. In this scenario, the traveling salesman's problem solving is the best way to deal with the shortest path through OR tools using SAT solver. The algorithm of dynamic programming helps in finding the best possible solution for the traveling salesman problem, in which the complexity of time exponentially upsurges with the increase in the number of cities (Cheikhrouhou et al., 2021).

INSTANCES, VARIABLES, AND FORMULATIONS

The SAT solver is competitive and it works best where there are side constraints. There is a sequence of multiple decisions which are used to implement the instances in SAT to determine the quickest path as well as the shortest pathway. SAT solver benefits to add additional restraints when the process is already implemented (Selsam et al., 2018).

In consideration of the linear programming formulations of the integers, the variables of SAT formulation comprise the cities and the distance between them. To be specific the different points of the cities are labeled as 1,…,n where, considering the distance, the formulation includes the time taken $c_{ij}>0$ to travel from city i to city j. That means the distance between the two cities should be greater than zero and the number of cities needs to be at least two. The variables specify the formulation which consists of:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

Figure: The variables and the formulation.

The total size of the formulation depends on the number of points that need to be determined. Since the approach is difficult, the number of functionalities is good in numbers, there is always a chance of change in dimension.

TRICK TO OPTIMIZE THE FORMULATION

The variables are only two comprising 0 and 1, which makes the formulation an integer program, whilst, the other programs and restraints are linear. Considering this situation, the main plan and trick that has been used are to minimize the length of the path that will be traversed. The optimized formulation includes:

$$\sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}.$$

Figure: The Optimized Formulation

Without more restraints, the $\{x_{ij}\}i,j$ will nevertheless efficiently array over all subsections considering the set of boundaries, that is very distant from the sets of boundaries in a tour, as well as consents for an inconsequential least distance where all $\{x_{ij}=0)$.

TIME REQUIREMENTS AND BOUNDARIES

The approach is a difficult approach to solve the traveling salesman problem. A formulation needs to be initiated to find out the shortest possible path to be traversed by the salesman. After finding out the shortest path, the minimum time-consuming point needs to be chosen and from that point, the entire process will take place making the minimum probable point to be the initial stage. Aiming toward this approach, a probability formulation needs to be determined to find out the possibility of the shortest path and the feasibility of the pathway as well as the formulation. Considering all these approaches, the entire process will be adequate and time-consuming.

There is a possibility to extend the process in the future. The number of cities needs to be more than two and the distance need to be more than zero. The formulation and the algorithm are robust in nature and it is expected that the outcome can be driven in a short time. Feasibility depends on the modeling and the approach. There are basically three types of approaches to deal with NP-hard problems. The 2-Opt programming method is very easy to deal with and the formulation can be of big sizes considering a greater number of cities as the points of the path. In this project, the Google OR-tools have been used which is considered to be the ultimate solution to solve NP-hard problems like the traveling salesman problem, Using the OR-tools benefited in

taking multiple and large numbers of inputs and it is feasible enough to deal with the problem as well as the formulation.

THE STANDARD FORMAT

The entire project is based on the Google OR tools and the two-dimensional matrix has been used. The array is implemented in the X and Y plane of the matrix. The array structure is taken and selected in such a way that it supports the formulation. The array needs to be the integers if the condition is in the integer, whereas the array needs to be full of character if the condition needs the characteristics. Both the data structure cases have been executed using the implementation case as well as the test case. The dimension can change according to the requirements and this standard format has been maintained.

The input format contains an integer N, which describes the number of cities or points that have been considered. The next test cases comprise the space-separated integers N, which includes DISTANCE [i][j]. The distance determines the distance between the city i to city j.

In our proposed tool we have taken a sample input like the following:

```
inp_data = {}
inp_data['dist_mat'] = [[0, 29, 15, 35], [29, 0, 57, 42], [15, 57, 0, 61], [35, 42, 61, 0]]
inp_data['total_vehicle'] = 1
inp_data['iteration'] = 0
```

Here we have been observed that in a various matrix we have taken the inputs. Executing this input information, we have found the following output like below:

```
Total Travel: 147 miles
Vehicle 0 routes:
 0 -> 2 -> 3 -> 1 -> 0
```

Where we have been observed that for the input the total travelled by the traveler and also with the defined vehicle the shortest route for travelling each of the city has been printed.

CONTRIBUTION OF THE DIMENSIONS

The distance matrix is an array in which the i and j denote the distance between the i location to the j location, wherever the array directories resemble the positions. The distance matrix is overtly distinct in the entire project. There is a possibility of using a function to compute the distances among the various locations. Aiming to illustrate, the Euclidean formulation is used for the distance among the points. Nevertheless, it is more effective to predefine all the distances amongst the locations as well as accumulate them in a matrix, moderately than computing them at running time. The two-dimensional matrix is researched to be quite advantageous for solving the traveling salesman matrix. Tsp [][] 2-D matric has been used considering each row has the array of the distances between the two cities from the initial point indexed stage to the next point.

CONCLUSION

Solving a traveling salesman problem is significant for many individuals who travel a lot in multiple cities owing to their jobs. Logistics and delivery services are deeply benefited through this outcome. The traveling salesman problem is an NP-hard problem that has been solved using the dynamic programming approach. The python-based SAT solver has been used to solve and satisfy the test cases. Considering the tools to solve the formulation, Google OR-tools have been used. All the tools and formulations used provides the best possible outcome to determine the shortest path from one point to another while keeping the traveling cost in check. There are some limitations and constraints in the project while taking the number of cities but considering the future possibilities, the research can be easily extended and more robust formulation and tools can be used to get the probable result in the shortest period of time (Chen et al, 2019).

REFERENCES

Cheikhrouhou, O., & Khoufi, I. (2021). A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. *Computer Science Review*, *40*, 100369.

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, *32*.

Dahiya, C., & Sangwan, S. (2018). Literature review on travelling salesman problem. *International Journal of Research*, *5*(16), 1152-1155.

Kim, M., & Park, J. (2021). Learning collaborative policies to solve NP-hard routing problems. *Advances in Neural Information Processing Systems*, *34*.

Kool, W., van Hoof, H., Gromicho, J., & Welling, M. (2021). Deep policy dynamic programming for vehicle routing problems. *arXiv preprint arXiv:2102.11756*.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., & Dill, D. L. (2018). Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.