\

TITLE OF THE PROJECT

Vehicle Routing Problem(VRP)

A COURSE PROJECT REPORT

By

C.KARTHIK REDDY(RA2111030010081)
K.LOKESHREDDY(RA2111030010105)
Y.SIVAMANI YADAV(RA2111030010073)

Under the guidance of
Ms.V.LAVANYA
In partial fulfilment for the Course

of

18CSC204J-DESIGN AND ANALYSIS OF ALGORITHMS

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District - 603203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR — 603 203

BONAFIDE CERTIFICATE

Certified that this project report “Vehicle Routing Problem” is the bonafide work of c.karthik reddy(81),k.lokeshreddy(105),y.sivamani(73)who carried out the project work under my supervision.

SIGNATURE	SIGNATURE
Ms.V.LAVANYA Assistant Professor Networking and Communications SRM Institute of Science and Technology Kattankulathur	Dr. Annapurani Panaiyappan K Head of the Department Networking and Communications SRM Institute of Science and Technology Kattankulathur

School of Computing

SRM IST, Kattankulathur – 603 203 Course

Code: 18CSC204J

Course Name: DESIGN AND ANALYSIS OF ALGORITHMS

Problem Statement	**Vehicle Routing Problem**
Name of the candidate	**C.KARTHIK REDDY**
Team Members	**K.LOKESHREDDY** **Y.SIVAMANI**
Date of submission	**4/05/2023**

Mark Split Up

S. No	Description	Maximum Mark	Mark Obtained
1	Exercise	5	
2	Viva	5	
	Total	**10**	

Staff Signature with date

PROBLEM STATEMENT:

The VRP is a generalisation of the travelling salesperson problem, a classic optimisation problem that has been examined since the 1930s. It isbased on a scenario where a salesperson must visit several cities. Their journey must finish back where they started, and they can visit each location only once. They can visit the cities in any order, but they need to travel the least possible distance. The problem is modelled as a network, with the cities represented by nodes and connected by paths weighted in terms of the time or distance required to travel between two cities.

DESIGN APPROACHES:

The vehicle routing problem (VRP) is a well-known combinatorial optimization problem that involves finding optimal routes for a fleet of vehicles to visit a set ofcustomers, subject to capacity and distance constraints.

Divide and conquer and back and bounce are two different algorithms that can beused to solve VRP. The time complexity of these algorithms depends on various factors such as the size of the input, the structure of the problem, and the specificimplementation of the algorithm.

DIVIDE AND CONQUER:

1. Divide the problem into a number of subproblems that are smallerinstances of the same problem.
2. Conquer the subproblems by solving them recursively. If they are smallenough, solve the subproblems as base cases.
3. Combine the solutions to the subproblems into the solution for theoriginal problem.

PSEUDO CODE:

Pseudo code for vehicle routing problem using divide and conquerapproach.

```
Function VRP_Divide_Conquer(D,C,K):
 //D:Distance matrix,C:customer demand ,K: number of vehicles
 //Returns a solution for VRP

//If there is only one vehicle,solve TSP for all customers
```

```
    If K ==1:
           Return Solve_TSP(D,C)


     //Divide customers into two groups

     M=len(c)//2
     C1 = c[:m]
     C2=c[m:]

    //solve sub-problems recursively
    Solution1 = VRP_Divide_Conquer(D, C1, K//2) Solution2 =
    VRP_Divide_Conquer(D, C2 , K-K//2)

    //Merge Solutions

  merged_solution = Merge_Solutions(solution1 , solution2)

   //Return merged solution

    return merged_solution
```

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


#define MAX 100

#define INF 9999999
```

```
// Structure to store the matrixtypedef

struct {

    int cost[MAX][MAX];int

    nodes;

} matrix;


matrix adj; // Adjacency matrix

bool visited[MAX]; // Array to keep track of visited nodesint best_cost

= INF; // Best cost found so far


void input_matrix() {

    printf("Enter the number of nodes: ");

    scanf("%d", &adj.nodes);

    printf("Enter the adjacency matrix:\n");for (int i

    = 0; i < adj.nodes; i++) {

        for (int j = 0; j < adj.nodes; j++) {

            scanf("%d", &adj.cost[i][j]);

        }

    }

}


int calculate_cost(int path[], int n) {int cost

    = 0;
    for (int i = 0; i < n - 1; i++) {
        cost += adj.cost[path[i]][path[i+1]];
    }
    return cost;
}


void print_path(int path[], int n) {
    printf("Path: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", path[i]);
    }
    printf("%d\n", path[0]);
}


void divide_and_conquer(int path[], int n) {if (n == 2)
    {
        int cost = calculate_cost(path, n);if (cost <
        best_cost) {
            best_cost = cost;
            print_path(path, n);
        }
        return;
    }
    for (int i = 1; i < n - 1; i++) {if

        (!visited[i]) {

            visited[i] = true; int

            temp = path[i];

            path[i] = path[n-1];

            path[n-1] = temp;

            divide_and_conquer(path, n-1);

            visited[i] = false;

            temp = path[i]; path[i]

            = path[n-1];path[n-1] =

            temp;

        }

    }

}


int main() {

    input_matrix();


    int path[MAX];

    for (int i = 0; i < adj.nodes; i++) {path[i] = i;

        visited[i] = false;
    }

    visited[0] = true; divide_and_conquer(path,

adj.nodes);printf("Best cost: %d\n", best_cost);

return 0;

}
```

TIME COMPLEXITY:

Divide and Conquer Approach: The time complexity of the Divide and Conquer algorithm depends on the number of recursive calls made and the time taken to merge the sub-solutions. In the worst case, the algorithm must make n recursive calls, resulting in a time complexity of **O(n log n)**, where n isthe number of cities. However, with efficient merging techniques and heuristics, the actual running time can be much less than the worst-case complexity.

BRANCH AND BOUND APPROACH:

Branch and bound is one of the techniques used for problem solving. It is similar to the backtracking since it also uses the state space tree. It is used forsolving the optimization problems and minimization problems. If we have given a maximization problem then we can convert it using the Branch and bound technique by simply converting the problem into a maximization problem. The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

PSEUDO CODE:

Vehicle Routing Problem using branch and bound pseudocode:

```
1. Initialize the search tree with the root node

2. While there are nodes to explore:

    a. Choose the most promising node to explore

    b. Generate child nodes by adding one city to the current path

    c. Calculate the lower bound of each child node

    d. Add the child nodes to the list of nodes to explore

3. Return the best solution found
```

Here is a more detailed pseudocode for the above algorithm:

```
1. Initialize the search tree with the root node
    a. Create a node with an empty path and level 0
    b. Set the lower bound of the node to 0
    c. Add the node to the list of nodes to explore

2. While there are nodes to explore:
    a. Choose the most promising node to explore
        i. Select the node with the highest priority (lowest bound)
        ii. Remove the node from the list of nodes to explore
    b. Generate child nodes by adding one city to the current path
        i. For each city not in the current path:
            1. Create a new node with the new city added to the path
            2. Set the level of the node to the level of its parent + 1
            3. Calculate the lower bound of the node
    c. Calculate the lower bound of each child node
        i. For each child node:
            1. Calculate the cost of adding the next city to the path
            2. Calculate the minimum cost of adding the remaining cities using agreedy algorithm
            3. Set the lower bound of the node to the sum of these costs
    d. Add the child nodes to the list of nodes to explore
        i. Add each child node to the list of nodes to explore, sorted by priority(lowest bound)
3. Return the best solution found
    a. When there are no more nodes to explore, return the best solution found(the node with
```

the lowest bound)

CODE:

Implementation of the Vehicle Routing Problem using Branch and Bound in C:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define MAX 100

#define INF 9999999


// Structure to store the matrixtypedef

struct {

    int cost[MAX][MAX];int

    nodes;

} matrix;


// Structure to store the partial solutiontypedef

struct {

    int path[MAX];int

    level;

    int cost;


} partial;


matrix adj; // Adjacency matrix


partial best; // Best solution found so far


bool visited[MAX]; // Array to keep track of visited nodes
```

```
void input_matrix() {


    printf("Enter the number of nodes: ");
    scanf("%d", &adj.nodes);
    printf("Enter the adjacency matrix:\n");for (int i
    = 0; i < adj.nodes; i++) {
        for (int j = 0; j < adj.nodes; j++) {
            scanf("%d", &adj.cost[i][j]);
        }


    }


}


void initialize() {


    for (int i = 0; i < MAX; i++) {
        visited[i] = false;
    }


    best.cost = INF;
    best.level = -1;
}


void print_path(partial p) {
    printf("Path: ");
    for (int i = 0; i <= p.level; i++) {
        printf("%d ", p.path[i]);
    }


    printf("%d\n", p.path[0]);
    printf("Cost: %d\n", p.cost);

}

void copy_solution(partial src, partial *dst) {dst->level
    = src.level;
    dst->cost = src.cost;

    memcpy(dst->path, src.path, sizeof(src.path));

}

void add_to_path(int node, partial *p) {p-
    >level++;
    p->path[p->level] = node;

    p->cost += adj.cost[p->path[p->level - 1]][node];

}

void remove_from_path(partial *p) {int last =
    p->path[p->level];
    p->cost -= adj.cost[p->path[p->level - 1]][last];p->level--;
}

void branch_and_bound(partial p) {if (p.level
    == adj.nodes - 1) {
        p.cost += adj.cost[p.path[p.level]][p.path[0]];if (p.cost <
        best.cost) {
            copy_solution(p, &best);
            print_path(best);
        }

        return;

    }

for (int i = 0; i < adj.nodes; i++) {if
        (!visited[i]) {
            add_to_path(i, &p);
            visited[i] = true;


            if (p.cost + adj.cost[p.path[p.level]][i] < best.cost) {
                branch_and_bound(p);
            }


                remove_from_path(&p);


            visited[i] = false;

        }

    }

}

int main() {
    input_matrix();
    initialize();
partial start;
    start.level = 0;
    start.path[0] = 0;
    visited[0] = true;

  branch_and_bound(start);

  printf("Best solution:\n"); print_path(best);

 return 0;

}
```

This implementation uses a recursive function called branch_and_bound() toexplore all possible solutions using a depth-first search strategy. The functiontakes a partial solution as input and generates all possible children by addingunvisited nodes to the path. It then recursively calls itself on each child and prunes branches that have a cost greater than the best solution found so far.The best solution is stored in the best variable.

TIME COMPLEXITY:

In the branch and bound approach, the problem is solved by iteratively improving an initial solution. The time complexity of this approach depends onthe number of iterations required to find a good solution, the size of the searchspace, and the efficiency of the heuristics used to guide the search. The time complexity of the back and bounce approach for VRP is typically **O(n^2)**, where n is the number of customers.

COMPARISON:

Both divide and conquer and branch and bound are common techniques usedto solve optimization problems like the vehicle routing problem (VRP).
However, the best approach for VRP depends on the specific probleminstance and the desired outcome.

Divide and conquer is a method that involves breaking down a problem into smaller, more manageable subproblems that can be solved independently. Inthe context of VRP, this could involve dividing the problem into smaller subproblems, such as dividing the routes into smaller segments or breaking the problem into smaller geographical areas. The advantage of this approach is that it can lead to faster computation times, as the smaller subproblems canbe solved more efficiently. However, it may not be suitable for all instances of the VRP, as dividing the problem may not always lead to a feasible solution.

On the other hand, branch and bound is a method that involves systematicallyexploring the space of possible solutions by branching out from the current solution and pruning branches that cannot lead to an optimal solution. In the context of VRP, this approach involves generating and exploring a large number of possible solutions, and eliminating those that are infeasible or suboptimal. The advantage of this approach is that it can lead to more accurate solutions, as it systematically explores the entire solution space.
However, it may be computationally expensive and may not be suitable forlarger VRP instances.

In summary, both divide and conquer and branch and bound are viable approaches for solving the VRP, and the best approach depends on the specific problem instance and desired outcome. In practice, a combination ofthese approaches, along with other heuristics and optimization techniques, may be used to solve VRP efficiently.