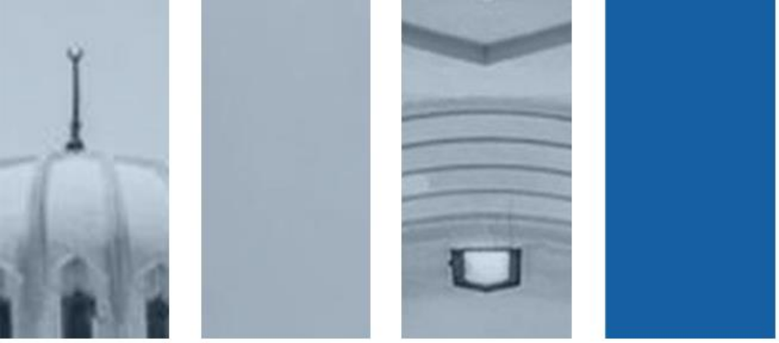


RESEARCH PROJECT SUBMISSION

Program: Computer Engineering and Software Systems

Course Code: **CSE 224**

Course Name: **Design and Analysis of Algorithms.**

Examination Committee

Dr. Gamal A. Ebrahim
Dr. Mohamed A. Taher
Dr. Mahmoud I. Khalil

Ain Shams University
Faculty of Engineering
Spring 2020 Semester

AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

CSE 224, DESIGN AND ANALYSIS OF
ALGORITHMS, SPRING 2020

Student Personal Information

Student ID:	17P8229
Student Full Name:	Ahmed Essam Mohamed Ramzy

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student **Ahmed Essam Mohamed** **Date:** **Fill in 5-6-2020**

Submission Contents

01

First Topic

Contents

I) ABSTRACT

This paper will show meaning of an optimization method, its goal what are their types by selecting four of the most famous methods used in different applications in engineering and industry. In this paper, we also will try to apply these methods to a chosen problem which proofed to be NP by comparing between the different methods and techniques from the complexity point of view by using tables and functions when needed and discussing when it is better to use a specific method and also by comparing the three approaches for any algorithmic approach.

A) *keywords:*

Optimization methods; Dynamic Programming; Travelling salesman problem; Brute Force; Divide and conquer; Greedy; Genetic algorithm; Particular swarm optimization; Branch and bound; Time complexity.

B) *introduction:*

No doubt, leaders and pioneers assessed that from the most significant procedures in any innovative technological design is the analysis and optimization.

Analysis Which Is a procedure wherein analytical tools, mathematical models and scientific standards got from the discrete sciences, mathematics, and from the discrete sciences, mathematics, and engineering fundamentals are utilized to build up a model item model that helps us to get an actual product [1].

While Optimization is the procedure to discover the best solution of a specific issue by following a few of conditions and assumptions that will give us either the most extreme or the base of a function exposing to the given constraints.

Nearly in all uses of engineering and industry, our motivation is to optimize something variable, Either by limiting the cost, time, space, energy consumption, Or by boosting the profit, effectiveness, performance, or even the optimal output of the given information by taking into consideration our given constraints.

Before diving into our optimization, let us clarify our goal of optimization methods.

C) *Goal of Optimization Methods:*

The objective of optimization methods is to locate the optimal or less-optimal solution with the minimum computational effort, This effort of an optimization method can be assessed either by the complexity of time and space and by the balance between the output of the solution and effort made[2].

II) OPTIMIZATION METHODS CATEGORIES:

A. **Exact optimization methods** : Such methods can ensure locating the ideal methods and leading optimization solutions where we have no assurance that the ideal solutions found, Typically, it is the option where an optimization problem can be solved with effort that rises polynomially with the period of the problem, As for example(branch-and-bound, branch-and-cut, cutting plane, and dynamic programming algorithms).

B. **Heuristic optimization methods**: Which overwhelms the NP-Hard at which the optimization approaches are problem-specific as they use the problems defined properties. Where they exhibit positive returns for many NP concerns and the problems of the same condition.

After discussing the types of optimization methods, we will dive into our optimization methods.

III) OPTIMIZATION METHODS:

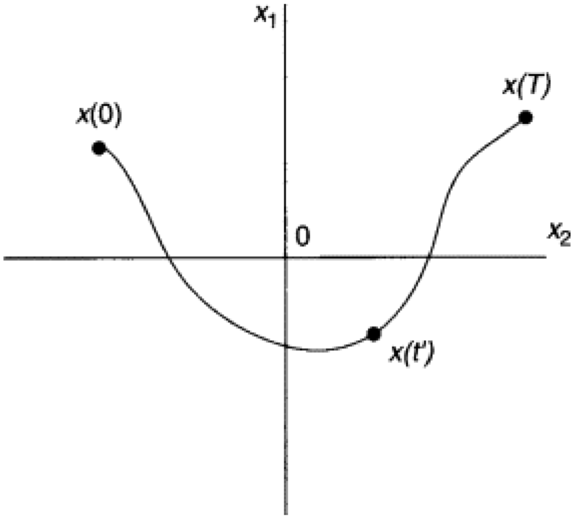
A)Dynamic programming:

1) Theoretical Foundation*:*

Dynamic programming is an optimization method that depends on the **standard of optimality** founded by the researcher Bellman where he expressed its definition, that for every ideal policy it is just comprised of ideal sub-polices. In dynamic programming, we will likely locate every conceivable solution that give the (maximal or negligible) estimation of the required. By multi-stage solutions, structure over which comprises various advances. Or, else a huge problem is partitioned into a succession of sub-issues. Furthermore, the word “Programming” in dynamic programming is to make the best decisions, while the word "dynamic" indicates the algorithmic sequence of operations and methods.

Principle of optimality of bellman:

If there is a path in the space that is between the beginning start x(0) and end x(T) points and is ideal in our perspective on any cost functional(from time, space,…) then the sub-path, interfacing any point x(t') of a similar way with a similar endpoint x(T) ought to likewise be ideal.



2) How it works:

Dynamic Programming is, for the most part, an optimization strategy running over the standard of recursion.Where on the off chance that we see a recursive method or procedure that is repeated for the same input.

We can optimize these costly calling methods using the Dynamic Programming approach, by storing the outputs and the significant data of the called functions, with an intention that we don't need to re-calculate them if we need them later. This can just lessen and advance time complexities from exponential to polynomial.

There are two ways for storing and reusing the values of the subproblems:-

a)Tabulation: It is a technique where we start from the base and continue saving the answers to the top(Bottom-Up approach), Let's take an example, Our AH[x], where its initial state is AH[0] So, we have to discover the estimation of goal state AH[n].

If we began our progress from base state AH[0]and proceed with our change of states to arrive at goal state AH[n] It is the Bottom-Up approach.

Example.

Let's discover the fact of a number by using tabulation [bottom up] approach

```
int AH[MAXSIZE];


int AH[0] = 1; //base state
for (int j = 0; j<n; j++)
{
   AH[i] = AH[i-1] * j;
}
```

The above algorithm clearly follow the bottom Top method starting the states from AH[0] and reaches its final state AH[n]. Here, we may notice why it is called tabulation as the AH table is being populated sequentially in sequence and we are reusing the calculated states from our table itself to diminish the expense of recalculating again.

2) Memoization Method

Once, again let's explain it in our state transition. If we want to discover AH[n] and instead of starting from the base state as in tabulation AH[0], We store the most updated values up to our final state so we return it from the memory. As this approach called memoization[3].

We can solve our factorial problem

```
int AH[MAXSIZE]


int fact(int i)
{
   if (i==0)
      return 1;
   if (AH[i]!=-1)
      return AH[i];
```

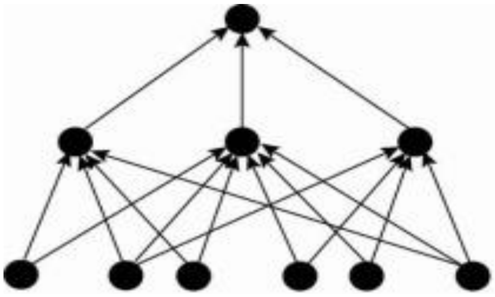
```
    return (AH[i] = i * fact(i-1));
}
```

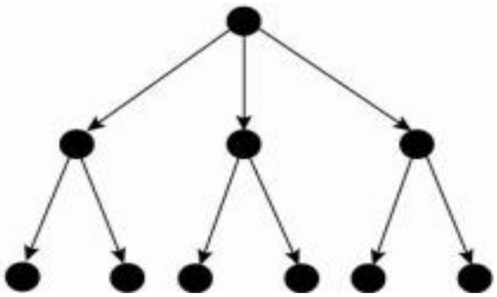
As we in the top-down approach the idea is to memoize and reuse using the recursive calls, but it is an inefficient, recursive algorithm[3].

3) When to use:

When you see an issue of optimal substructure in which you can divide a huge problem into simpler and smaller portions and afterward consolidate them to get the ideal solution, Or by another way if you are using recursion; Here you could think of the divide and conquer approach, But also whether the subproblems are overlapped with each other, I mean by overlapping where we recalculating the same thing a lot [4].

So when you approach a problem that has a property of,maximization, optimization or counting i.e shortest path which is a minimization problem or maximizing the profit then this is a property of dynamic programming, As for the shortest path you must pass all over the possible solutions and find the best of them.As explaining the substructure solution we find that there is a similarity between divide and conquer and dynamic programming as both of them depends on the sub-problem of the previous solutions. and both of them depend on the recursion method. In divide and conquer approach the divided sub-issues are utilized together into forming the large needed problem, unlike in dynamic programming where storing the results of the sub-problems to use them to get the larger sub-problems.





Dynamic Programming

Divide and conquer

4) Global convergence to the optimal solution:

An issue considered to apply the rule of optimality is that if the sub-issues solutions of a perfect solution of the problem are the equal perfect ideal solutions for their sub-problems, If our optimal solution can be divided into a set of sub-problems solutions, optimal substructures need that those sub-solutions must be optimal with their consecutive sub-problems.

To indicate a problem to be an optimal substructure, we should demonstrate that it has the following two properties *substructure* and *optimality.*

Dynamic programming uses the method of recursion, Where the problem is a **substructure** if it is can be subdivided into subproblems and the subproblems also could be divided until reaching the trivial solutions (i.e constant)Where dynamic programming depends on the **Optimality** *property* where the summation of the optimal solutions to the smaller subproblems will make a globally optimal soultion[4].

5) TSP using the dynamic programming:

a) Problem domain:

TSP problem is that to find the least cost for the salesperson to travel a bunch of cities with the least time given a matrix of distances between n numbers of cities, with a constraint that the given cities in the tour must be explored at least one time. and for solving any problem using the dynamic programming, the problem must implement these basic three elements

- Substructure: We divide the given problem into smaller and simpler and mini-problems.
- Table-Structure: After having these sub-problems we save the results in a table to actually reuse them again for minimizing the computational overhead.
- Bottom-up calculation: Using that table we use the solutions of that sub-problems for the sake of solving the larger problem.

And the TSP actually use them for solving the problem so TSP is from the domain of problems that get solved by dynamic programming.

b) Limitations:

As said Dynamic programming can be applied to the *standard of optimality*, This infers that partial sub-problems can be contacted to the state after the partial solution as opposed to the incomplete solution itself. To conclude whether to expand an inaccurate string organized by a substitution, addition, or eradication, we do not need to know absolutely which arrangement of operations was performed to date.

Our limitation on using dynamic programming is the number of partial solutions of the sub-problems that we must log. The partial sub-problems can be consisting of the stopping positions in the input. This is because the objects took a shot at (strings, numerical sequences, and polygons) all have a verifiable request characterized upon their components. This request cannot be mixed without totally changing the problem[5].

We have an exponential number of conceivable partial solutions and are bound to require an infeasible space of memory.

In dynamic programming also, its hard to write code that obtains the best efficient order for the sub-problems and of getting a good arrangement of solution.

c)Explanation of code:

Tsp has several usages in many different applications and as it is considered to be in a class of NP-complete problems, but it is efficient algorithm for calculating the TSP problems as running time increases with an exponential factor with the total number of cities, so if the number of cities instances reached hundreds it will take years for our machine to solve it, this problem also can be modeled as a complete graph which that all two connected cities is linked by an edge but with a problem that instances of such not connected cities can be switched to complete graphs of very large edges that will not be discovered in our ideal tour whose vertices mapped to cities and the edges mapped to the path between these cities.

There is two types of traveling salesman problem :-

- Symmetric TSP: If the distance between the two cities is just precisely the same in each direction, the number of successful solutions is halved.
- Asymmetric when in other ways the distance from the city to the other is not equal.

Our goal is to find least tour, which fulfills all the traversal of the salesman by enumerating over the three phases:-

1) We go for the least calculated paths for the person starting from the first one passing by the end one heading back to the first again.
2) Setting the efficient functions of the least costly route to cover these nodes.
3) We finalize by storing the best locations traversed.

Let’s assume 5 cities (1,2,3,4,5) and there is a salesman who will tour each city starting from city 1 and returning back to the same city, As its overhead to visit the same city more than once so our problem is to complete the tour with the least cost of the edges.

As for city (1,{2,3,4}) the person at city number 1 initially and he has the option of moving to either city of {2,3,4} and as in dynamic programming the large problem is divided into a bunch of sub-problems, and we keep for this sequence till the end.

Assume that this table is the cost of the edges the person traverse.

#	1	2	3	4
1	0	4	1	3
2	4	0	2	1
3	1	2	0	5
4	3	1	5	0

After solving the example we could write the equation

Recursive Equation

City(i , A) = Min ((i , j) + City (j , A– { j })) ; S!= Ø   ; j € S ;

A is set that has the number of unvisited cities.

City (i, A) : We are traversing the vertices from vertex label i to all the cities of set A of unvisited cities and go back to vertex 1.

Min (i, j) is cost of passing from i to j.

Also we don't guarantee that each vertex is such connected to the neighboring vertex so the cost will be infinity and then we replace it with the minimum and all the paths which still infinity won't be calculated .

d) Complexity

As dynamic programming lies on principle of sub-problems and for the nodes i and solving the recursion equation.

$$T(n)= \sum_{V=1}^{n-1}(n-1-v)v\binom{n-1}{V} = (n-1)\sum_{V=1}^{n-2}V\binom{n-2}{V} = (n-1)(n-2)2 \wedge n-3$$

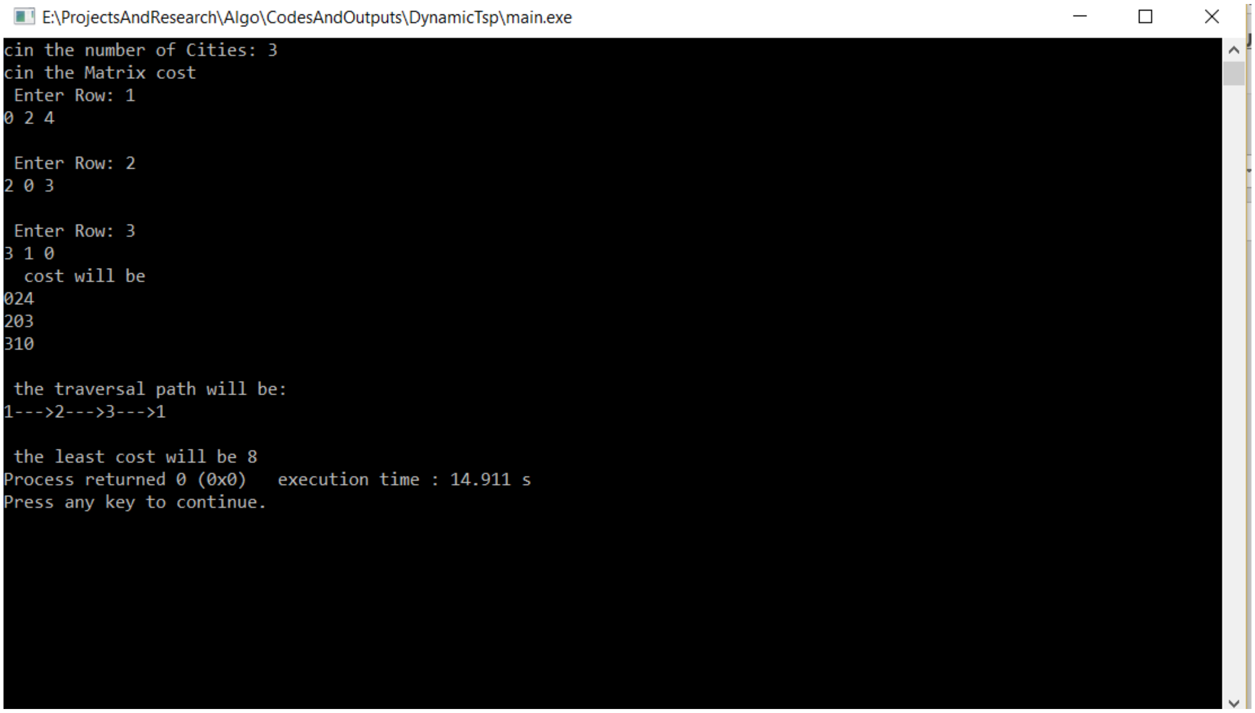
So time complexities :-

1.Big-O: O($N^2 2^n$)

2. Big-Θ: Θ O($N^2 2^n$)

3.Big-Ω : Ω O($N^2 2^n$)[6].

e)Output of dynamic programming:



E:\ProjectsAndResearch\Algo\CodesAndOutputs\DynamicTsp\main.exe

```
cin the number of Cities: 3
cin the Matrix cost
 Enter Row: 1
0 2 4

 Enter Row: 2
2 0 3

 Enter Row: 3
3 1 0
  cost will be
024
203
310

 the traversal path will be:
1--->2--->3--->1

 the least cost will be 8
Process returned 0 (0x0)   execution time : 14.911 s
Press any key to continue.
```

B) Brute Force approach:

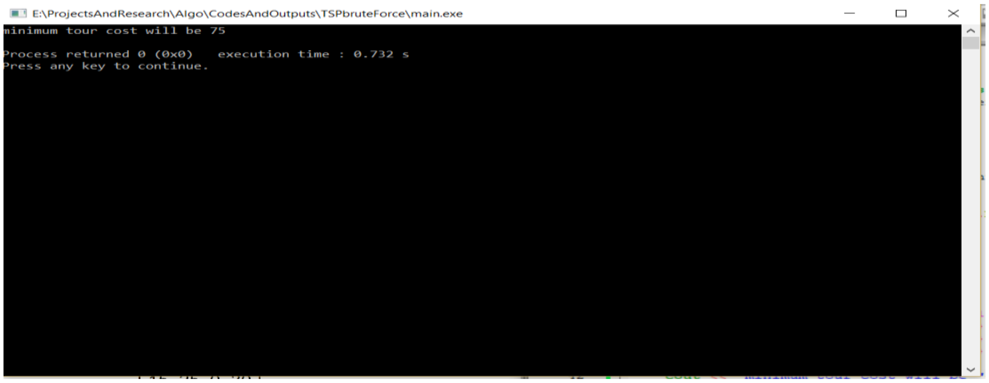
Explanation: We will utilize a naïve brute force as will assign the starting city by 1 and ending city 1 also and we will calculate for each iteration and keeping track of the minimum cost we will use the Hamilton(there will be a tour that goes on every city only one time) cycle by considering that our graph is complete.

1)Complexities:

I calculated the weight the hamilton cycles by adding wights of its edges

So its time complexity is Big O ((N −1)!)

2) Output



E:\ProjectsAndResearch\Algo\CodesAndOutputs\TSPbruteForce\main.exe

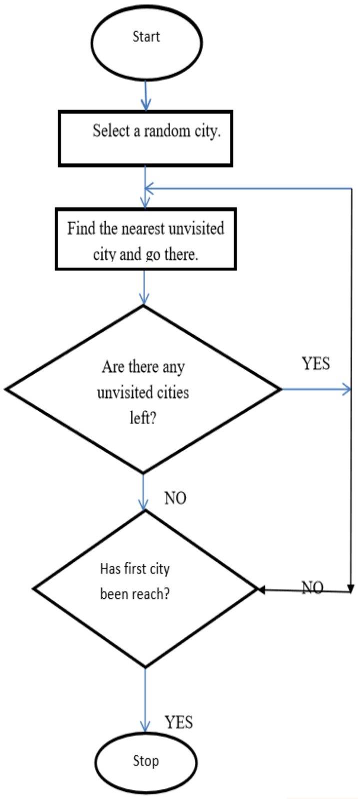
```
minimum tour cost will be 75

Process returned 0 (0x0)   execution time : 0.732 s
Press any key to continue.
```

c) Greedy approach:

We will solve the Travelling salesman issue using the **nearest neighbor** algorithm, the salesman begins at any city and will visit each nearest city until he visits them all.

Our algorithm will be as follow we will initialize all vertices (cities as unvisited), then we select a random vertex(city) we will calculate using the cost() the shortest edge connecting this vertex using a greedy approach and we will do this for all the cities until they are all visited then we will terminate.

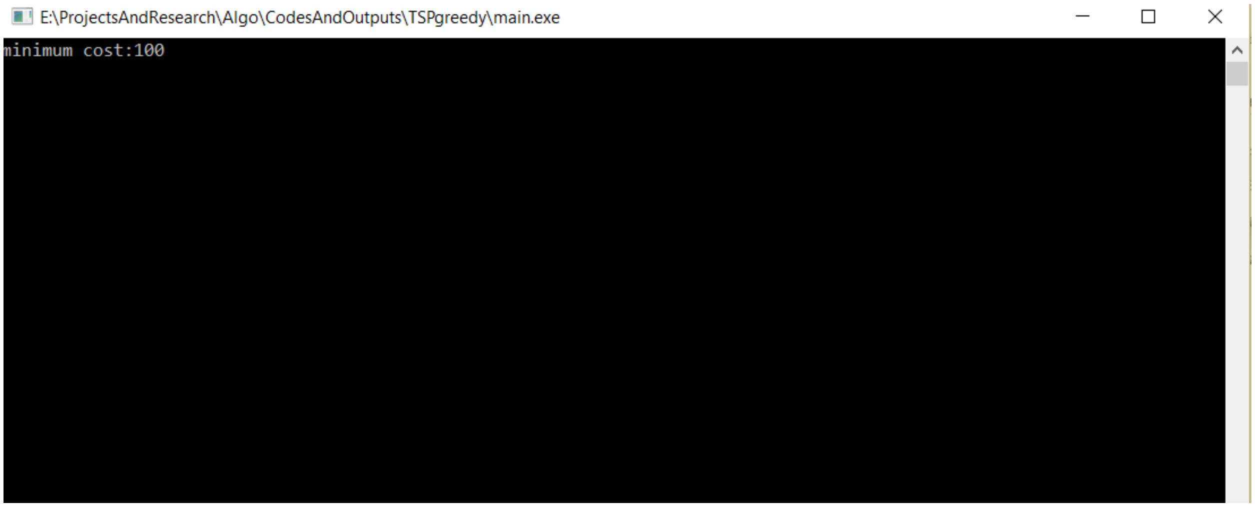
The Nearest neighbor algorithm is such an easy fast but it can miss shorter routes as its greedy approach [7].


Start
Select a random city.
Find the nearest unvisited city and go there.
Are there any unvisited cities left?
YES
NO
Has first city been reach?
NO
YES
Stop

1) Complexities:

Time complexity in big O will be of O($N^2 * \log$ (N)) at which the N is the total number of cities that are visited and there is also another greedy method that is somehow heuristic which searches for the optima that starts with arranging all the edges and then choose the one with least cost and continue selecting for the best next selections but also with a big O($N^2 * \log(N)$) [8].

2) Output



E:\ProjectsAndResearch\Algo\CodesAndOutputs\TSPgreedy\main.exe

```
minimum cost:100
```

D) Divide and conquer approach:

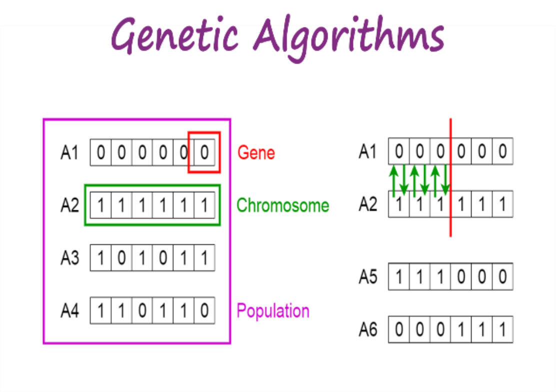
As there are various algorithms and local searches try to solve the problem, But these class of problems are difficult as they have not such an easy structure and as this approach depends on evaluating the problems into subproblems and summing them together so there is no such an approach divide and conquer approach for the traveling salesman problem[9].

E) Genetic Algorithms optimization method:

1) Theoretical Foundation*:*

(GA) is heuristic inquiry algorithms which is a one of the huge methods that used to solve combinatorial improvement issues and one of a huge class of evolutionary algorithms(EA) which in turn a huge field in computer science and research , the ga utilized for top-notch solutions to a tremendous issues of optimization and in searching, It is based on the ideas of genetics and selection (Where the species will be selected and go to next generation where they can adjust and survive to the changes in their condition).

The source of inspiration came from the process of evolution, Each generation may contain a populace of individuals where each can be an array of (ch,int,float,bit) can be mapped to the related chromosome. So GA is population-dependant algorithm[10]. It starts with the scientist John Holland's with his Schema Theorem was the first mathematical model of how genetic algorithms works[11]. Where his proposed gene could have one of three: (Zero, 1, do not care).



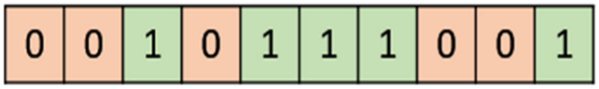
The inspiration came from a biological comparison for the GA where GAs works same way as human behavior, As the human works where each individual have a fitness level where it maps how good the solution to solve the problem . As the highly fit persons are given more chances to reproduce with others, this produces new “offspring” ,Where the low fit persons will have less chances for recreation and “die out”.

2) How it works:

A GA needs:-

Genetic representation:

This representation for facilitating the crossover representation, where it is proofed that bad representation can prompt horrible performance of the Genetic Algorithm. Each representation have a problem constraints, while the most used representation is an array of bits [12].



Array of bits representation

Fitness Function : A fitness function defines fitness of the chromosome to make sure it will reproduce and survive for the following generation (It represents the probability of the survival), This function returns a score which represents how the chromosome that is mapped to an individual(how well it solves a problem)and its input is the candidate problem solution[13].

Note to be thought of while computing the fitness function it must be as quick as the moderate figuring of the fitness function will influence the GA in general and will make it moderate.

The fitness function and the objective function have the same goal as both of them need to limit or augment the given objective function .With combinatorial optimization there are performance measures we want to optimize (cost,time,width,span ,…)

GA operation algorithm will be:-

1) Produce an initial populace as there is two methods for initializing:-

I) Random Initialization – Populate it randomly.

II) Heuristic initialization − Populate it in a heuristic way.

So the better is the random population as the heuristic will bring about comparable solutions.

There two things to take care

1. The size of the population, came from multiple theoretical ideas, as there should be some optimal value for the string given by taking care of the trade-off between effectiveness and efficiency. As for too small a population wouldn't be a suitable room for search space. Therefore, the population size will increase as an exponential function with the size of the string [14].

2) Fitness Function: calculate each chromosome by estimation of the fitness function.

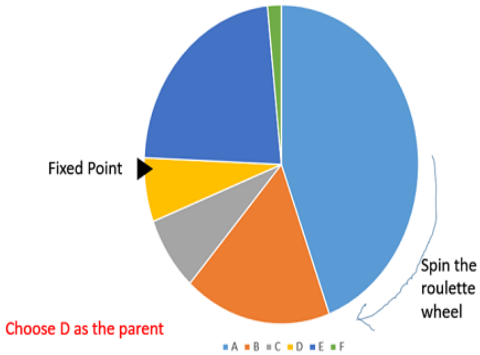
3) If termination criteria not satisfied then

4) New populace: Generate another populace by assessing

a. **Selection**: Procedure of choosing which parents to mate and its a significant procedure where we ought to keep up a decent diversity where one fit arrangement cant take the whole populace cant let one fit arrangement. It additionally ought to be related to fitness. I will examine two of the implementation of fitness selection

I) **Roulette Wheel Selection**

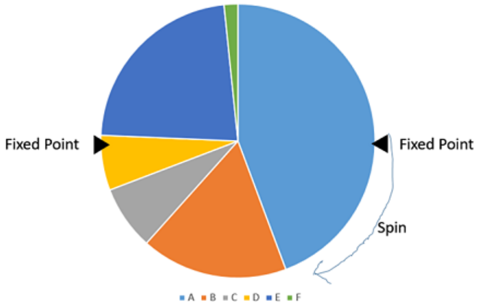
In this selection, the wheel is portioned and its conspicuous clear that the one who has prominent pie has a more noteworthy possibility of positioning aside from the fixed point. the parent is chosen if it comes in the locale of the fixed point. A similar procedure occurred for the other parent.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

II. Stochastic Universal Sampling (SUS)

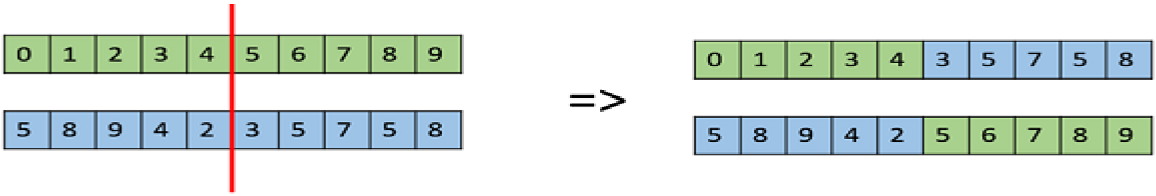
It is like a roulette wheel, but not having one fixed point will have m different focuses So, all the parents are chosen in one turn of the wheel. [15]



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

b. **Crossing Over**: The crossover is the way of making a superior fitness in the resulted offspring by blending the genetic representation of the parent populace individuals, more than one parent is picked and produce more than one offspring's need to cross in some way or another. There are more than one crossover administrator so the GA designer must choose explicitly to the problem space.

- One Point Crossover



0 1 2 3 4 5 6 7 8 9

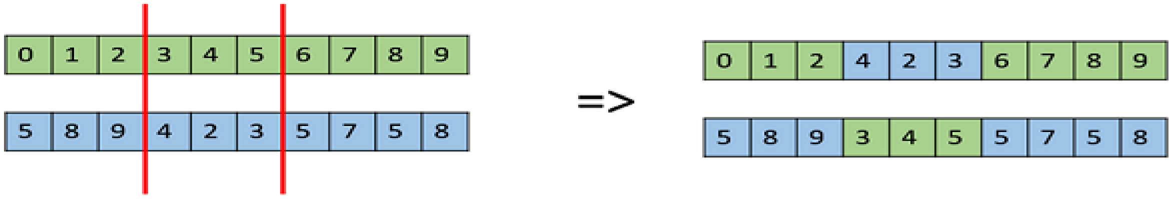
5 8 9 4 2 3 5 7 5 8

=>

0 1 2 3 4 3 5 7 5 8

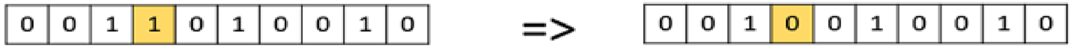
5 8 9 4 2 5 6 7 8 9

- Multi Point Crossover

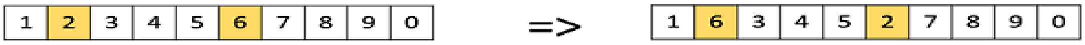


C .Mutation: Is the procedure to adjust at least one value of genome in a chromosome to keep up a genetic variety from this population generation In this procedure solution may change totally from the precedence one. Hence Genetic Algorithm gets improves solution from mutation. The probability of the mutation in the not at all like in cross over, Also it prevents the likeness of populace in chromosomes[16].

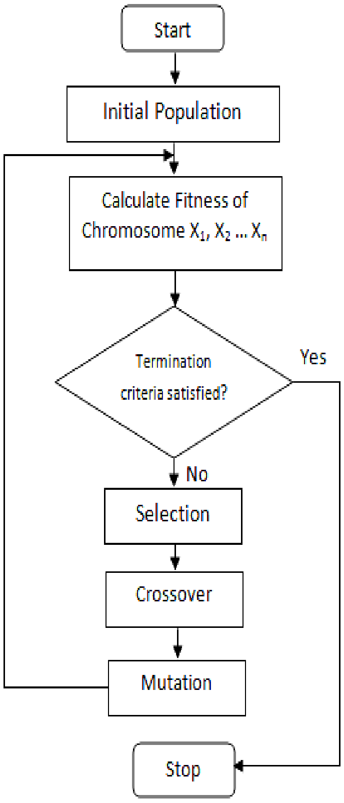
- Bit Flip Mutation.



- Swap Mutation randomly



5. On the off chance the termination criteria is fulfilled then finish, creating how «better solution» (not befuddling like optimum solution) and proceed.


Start
Initial Population
Calculate Fitness of Chromosome $X_1, X_2 \ldots X_n$
Termination criteria satisfied?
Yes
No
Selection
Crossover
Mutation
Stop

3) When to use:

To use genetic algorithms for a problem or not is by noting the question; what is the space you looked for? If the looked through space is unstructured and uncertain then GA representation of the created space will be extremely effective used for enormous space, if the search techniques is for certain reasons then GA will not be that proficient.

Likewise, it is smarter to use GA where the fitness function is ready or can be evaluated; As the GA’s are acceptable to look through the inquiry space by using the crossover process explained before. they are effectively circulated and to discover global optimum without being stuck in nearby optima[17].

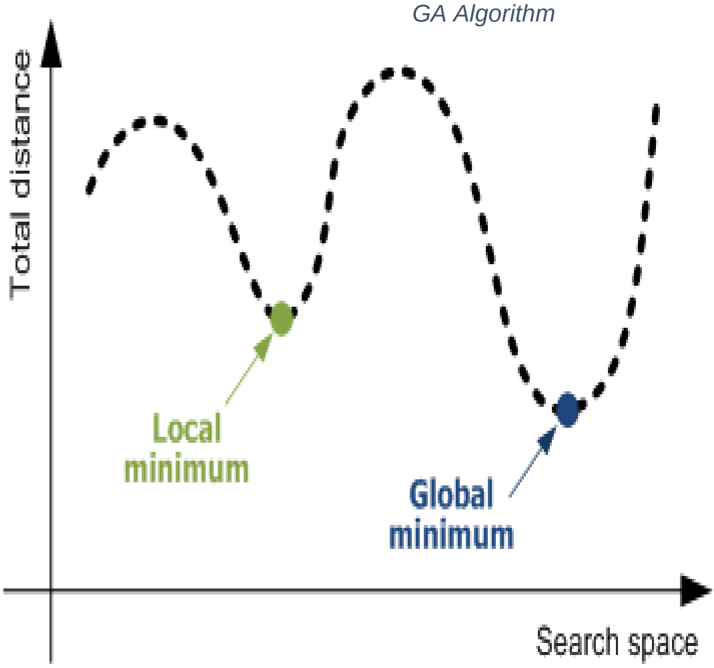
4) *Global convergence to the optimal solution:*

For keeping away the local strength and to converge away from the global solution where such a tradeoff between investigation and utilization. The acceptable results between exploration and utilization will prompt expanded the Genetic algorithm performance.

We can keep the best solution in the populace in each time of the procedure with the exception of the fittest or the first class, This guarantees that the inquiry doesn’t separate to a solution that has higher value of the objective function.

So as to ensure that the search continues until the predefined number of solutions, If the populace comes into a single solution so the search is re-initialized, this union is estimated by the genotype or by the objective function.

GA Algorithm



Where is the convergence parameter, f (Max) is the maximum objective function in this populace and f(Min) is the best objective function in the population[18].

Convergence function

$$\alpha = \left(\frac{f_{\max}(\cdot) - f_{\min}(\cdot)}{f_{\min}(\cdot)} \right) \times 100$$

5) Traveling salesman problem using the genetic algorithm

a) Problem domain

Traveling salesman problem is one of the famous optimization problems discovered, you have a bunch of cities and you wish to discover the tour associated these cities by at visiting every city just a single time and return back to the starting tour city and we want to do this tour in a limited manner. By given a graph G = (V, E) where it's weighted on the edge by the separation between cities.

We will have an assumption, For the city it will be genes, String formed will be chromosome, the Main fitness function is from the path of the length of cities given; also the less of the score of fitness is the fitter of the gene.

The fitness will be scored according to the gene path. As the path is lesser the more fit is the gene. As the gene is fitter than the other as the probability of surviving is higher. The fittest will survive and moved to the next loop. The number of loops depend on the cool function () and will stop in after some iterations.

TSP problem could be solved by numerous techniques (Exact algorithms, approximate algorithms),One of the strategies to discover the optimal solution is the exact algorithms through a various advances yet it will be of exponential complexity [19], It will not be an be a suitable strategy if the size of TSP becomes huge(100 cities for example).Therefore the approximate algorithms like the GA algorithms can be useful for NP-hard problems having short running time.

b) Limitations

For characterizing the representation for the TSP, the language must to be strong for determining the candidate solutions ,Also coding the fitness function as for a wrong answer it might prompt issues of getting the wrong solution. Genetic algorithms is not utilized for analytical problems.

The population used for evolution should be suitable for one problem (100).

Crossover rate should be 50%-70%.

Mutation rate ranging from 0.05%-0.09%.

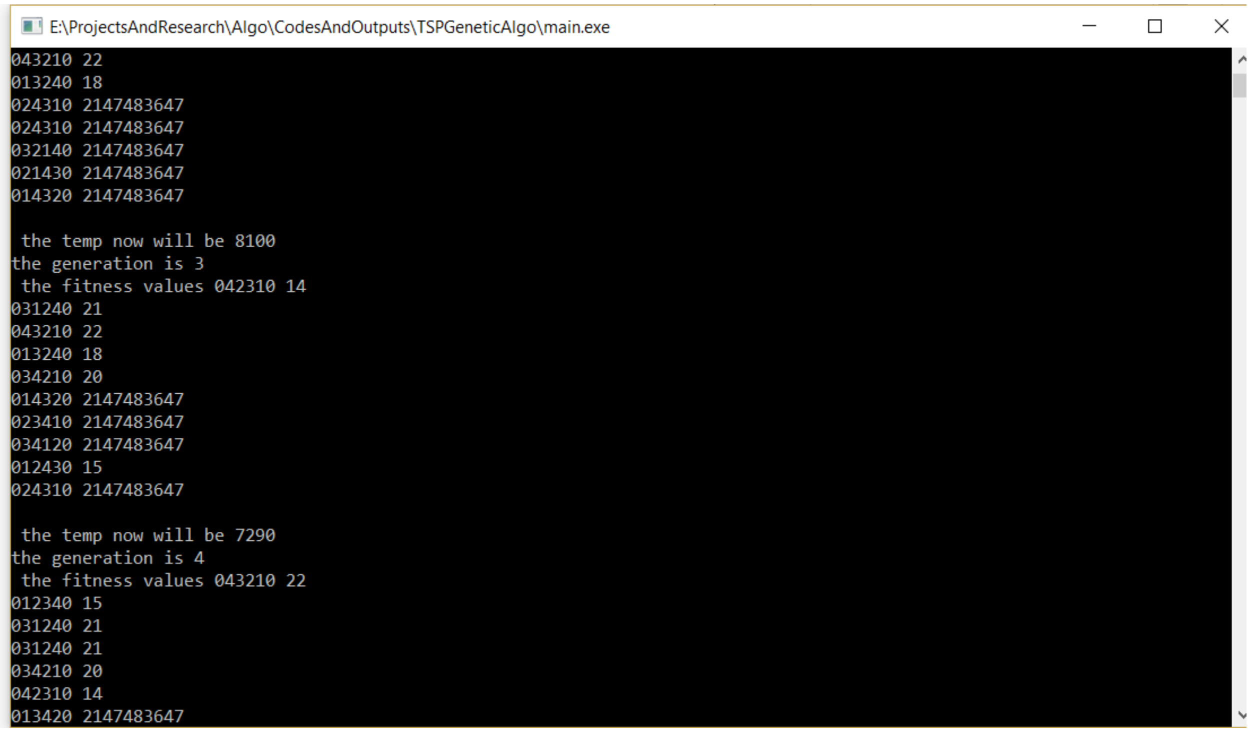
Suitable fitness function is designed

The method of selection should be appropriate [20].

Also for TSP the genetic operators lead to poor results by making too invalid solutions, leading As for TSP require new genetic operators and other representations like (Position Dependent Representations). Furthermore, GA can represent invalid solutions by make making wrong chromosomes; Occurs in arbitrary advance of initialization step of the GA's a mutation and crossing over. Two tours having the same cities in different starting points with the same order will be represented by 2 unique chromosomes, for example: tour (23541) = tour (12354).

c) Explanation of code:

We characterize the population size of 10, Then characterized the number of cities of 5 cities with names of genes ABCDE, Our genome structure that will map to an individual of 2 attributes a string path which maps to the path that the travel salesman traversed with an int fitnessVal, The function repeat will check() if the character has occurred in the string and mutation()method that returns a gnome with a random of two genes to create variation in species, And populize () to create the population and return a valid gnome string, fitnessVal() to return fitness value of a gnome , TSP() that initializes the variables of gen and populate the gnome pool and make the crossing and gene mutation.


E:\ProjectsAndResearch\Algo\CodesAndOutputs\TSPGeneticAlgo\main.exe

```
043210 22
013240 18
024310 2147483647
024310 2147483647
032140 2147483647
021430 2147483647
014320 2147483647

 the temp now will be 8100
the generation is 3
 the fitness values 042310 14
031240 21
043210 22
013240 18
034210 20
014320 2147483647
023410 2147483647
034120 2147483647
012430 15
024310 2147483647

 the temp now will be 7290
the generation is 4
 the fitness values 043210 22
012340 15
031240 21
031240 21
034210 20
042310 14
013420 2147483647
```

d) Complexity:

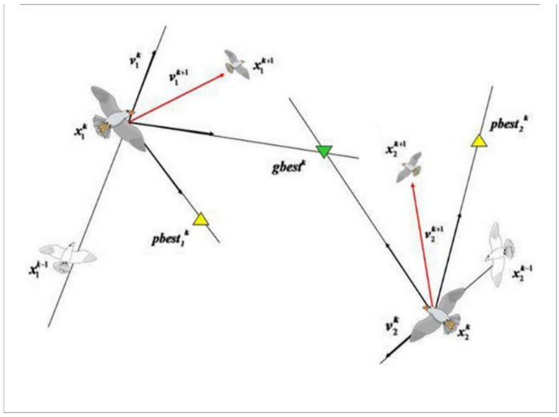
As we probably aware that TSP has a time complexity exponential, So if there is a number of cities n it will have a precise solution of O(n!) factorial time which is an expensive time to search for, our solution is genetic algorithms with its operations will give us a complexity of time of O(O*P*N^2) at which S is outer iterations , O is the size of the populace, N^2 for all repair techniques that search every pair of vertices necessary.

In any case, if we consolidate the GA with local algorithms for search and divide the time complexity will be O(SM$N^{2.2}$)[21].

F) Particular swarm optimization (PSO):

1) Theoretical Foundation:

In 1995 Kennedy and Eberhart proposed an algorithm for an optimization technique based on the swarm which defines a way of how any swarm gets a way to find food. It is a swarm intelligence, population-based method where population P={p1,….,pn} all of this population is considered a candidate solution and it is mapped to a swarm. In addition, it represents that state of the algorithm by a population, which is, continues modified until a termination after doing some studies on the behavior of animal swarms we found that some of them are able to share information that helps with their survival. The first theoretical analysis indicated in one dimensional PSO where there wave looked like a sine where a particle moved along this path[22].Another analysis method was a deterministic dynamic system depending on the random coefficients who results in a second-order dynamic systems that is a linear one[23].



Considering a swarm of birds searching food, we consider a search space where the population members are particles or birds mapped to solutions and velocity v which direct the flying of the particles, at each time there is two values can be updated to the particle the fitness function->pbest and the global best->gbest all of these will be expressed in Equation(1) [24] .

Equation(1)

V[i](t + 1) = v[i](t) + c1. rand1.(pbest[i](t) – x[i](t)) + c2. rand2.(gbest[i](t) – x[i](t)

Equation(2)

present[*i*]=present[*i*]+*v*[*i*]

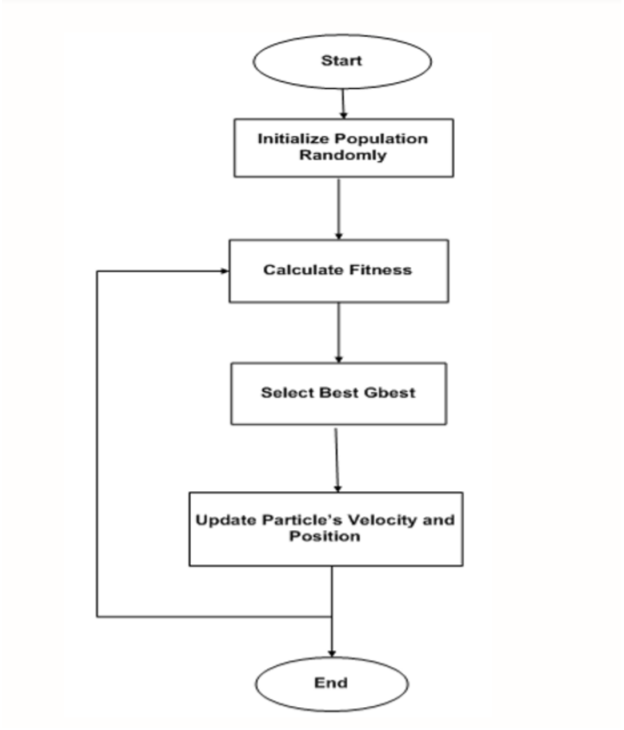
2) How it works*:*

PSO elements that we will use in the algorithm

- Particle: Our particle Pi that is represented for real numbers.
- Fitness Function: Our Fitness Function or the objective function where it is utilized to get the optimal solution.
- Local Best: The candidate suitable position of the particle for all the positions visited so far.
- Global Best: The position for all particles visited which the best fitness function is calculated

- Velocity : It is used to determine the direction and speed of the particle and is recalculated with Equation(1)
- Position Update: All of the particles that want to discover the optimal fitness by updating their positions as in equation (2).

```
1. Initialization
   For each particle I swarm size of population P
         Initialize Xi and Vi randomly
                  Calculate  fitness f(Xi)
                         Initialize Pbest(i)
                  Initialize Gbest keeping a copy of Xi with the
                  best fitness.
2. While (! termination)
   For each particle i:
     Update Vi by Eq(1) and Xi by Eq(2)
         Evaluate fitness F(Xi)
         Pbest->Xi if(f(Pbest(i))<f(Xi)
         Gbest -> Xi if(f(Gbest(i))<f(Xi).
```


Start
Initialize Population Randomly
Calculate Fitness
Select Best Gbest
Update Particle's Velocity and Position
End

So for the PSO algorithm given it tries to optimize the problem iteratively by improving the candidate particular solution by having a populace of particles in the search area and evaluating the equation 1 and 2 to update the position and velocity. For each particle, its movement will be constrained by the best method locally and try to be the global best in the given search area [25].

3) When to use:

As both PSO and GA is from the evolutionary techniques so the same procedure:-

1) An initial random population generated.

2) Calculating fitness value for each one will depend on the distance to the optima.

3) The fitness value will depend on the recreation of the population.

4) This will be in a loop and going back to step 2 until the termination criteria happen.

They both are similar in the population created at random and they evaluate the fitness function and they keep updating the population and search without guaranteeing any success for the optima, But PSO doesn't have operators(crossover or mutation)as the particles updated with their velocity Also for the sharing mechanisms, PSO is different from the GA as the population as a whole in GA chromosomes will send the important data with each other towards the optima, While in PSO the Gbest give the information to the other particles..

In GA values handle discrete problems as they can be changed into 0's and 1's , while PSO variables are dependent on their positions in the space so could take multiple values corresponding to the velocity vector.

In the GA it converges toward local optima rather than global unlike the PSO.

So we use the PSO as it only requires a few adjustments to adjust, and it checks for functions of local and global, Have memory to store the fitness previous values, And its implementation is not so hard.

4) Global convergence to the optimal solution:

The PSO algorithm converges as the swarm selected search and this convergence doesn't guarantee that the output is the optima or local optima, Until van den Bergh and Engelbrecht update an algorithm converges , It seeks the optimal global particle equation, which allows them to produce a random search around the optimum global location when the other particles are equated , it could cost a speeder rate of convergence. So from the factors that affect the convergence of PSO towards a local optima is the shortage of population.

Keeping the convergence feature of the PSO by getting away from the premature convergence on the local optima is an important reason why there is some variants in PSO are offered.

Such as the adjustment of the parameters of the PSO to update the particle's velocity that various PSO consider it the best solution within the next particle instead of the entire swarm and integrating another heuristic algorithms with the PSO [26].

As in PSO the early particle is transmitted during the PSO search is efficient for the optimization task of low dimension, however with regard to high parameter the algorithm must be stuck in the localized minimum and the Gbest may not change over many iterations; It is simple for the algorithm to evade the close minima so the essential PSO can meet quickly.

As the particle updates from one generation to another the particles of the swarm could calculate the mean dimension for number of particles calculated from equation where t is the generation number.

$Pdi = (xi1 + xi2 + \ldots\ldots + xiD) / D$

Where D is the dimension of the particles.

$V[i](t + 1) = w*v[i](t) + c1.\ rand1.(pbest[i](t) - x[i](t)) + c2.\ rand2.(gbest[i](t) - x[i](t)) + c3.\ rand3.(Pd[i] (t) - x[i](t))$

After introducing the above to the velocity formula, Pbest,Gbest of the next generation it is rising the amount of information available, So now finding the optimal solution is easy and the above formula weight coefficient is sligh compared to the disruption data which boosts the particle variety. pd[i] might transfer these particles to a better area can and reduce the Gbest attraction to the local minimum solution. [27].

5) TSP using the PSO method:

a) Problem domain

Although the PSO algorithm is effective in real applications and has scored powerful theoretical early findings results, but it really hasn’t provided sufficient evidence needed for the algorithm to converge. The actual issues get evaluated by the constraints in the algorithm. the easiest approach to work with the continuous variables is to use Discrete/binary PSO algorithm are best to deal with the continuous variables, But it is limited for the discrete variables. As with the particular problems of application we must analyze the PSO algorithm from the depth as well as width and to concentrate on the highly PSO design and adding it with the optimized problems and rules. As for the problems solved by the PSO algorithm, we should update the formula and the strategy to take care of the local and the global exploitation.

b) Limitations:

As for the local extreme functions, it can’t get the correct result because of the particles diversity that disappear quickly and the optimized functions that result in causing convergence premature and also due to lack of the suitable search methods, Also the algorithm don’t use the information from the output of the equation calculation procedure every iteration, It uses only the information provided by the swarm and individual optima. Also the PSO algorithm can’t guarantee the convergence to the optimal solution providing the potential outcome of search mechanism can not also assure the global optimal convergence. Since the PSO algorithm is an optimization algorithm, It is intended to ease for simplifying and simulate the search of certain swarms, though it don’t specify its range of searching So, PSO algorithm is generally suitable for high dimensional and not too accurate solutions lacking a mathematical foundation that helps in the future relevant theories [28].

c) Explanation of code:

PSO algorithm can be utilized to better solve the TSP as its easy to be implemented and having a small number of paramaters and that PSO is very good with the continuous problems .As for the TSP providing a graph A = (B,C), where A = {1,...,x} and C = {1,...,Y}, and costs p[r][q] as for edge i,j the TSP goal is to find the minimum total length of hamiliton cycle G by calculating by the total amount of the edge costs in the cycle in enquiry.

For all the nodes of {r,q} that costs of r,q and p of r,q if they are equal then it is a symmetric problem if not it is asymmetric.Adapting PSO where to consider the velocity operators concerning the movement of particles.

PSO algorithm have to be adapted as for the discrete problems regarding optimization. Kennedy & Eberhart suggests a distinct binary PSO version, which defines the paths and speeds of particles in terms of change in bit probabilities of (0,1)[29]

Pseudocode:

```
method PSO
 {
Instantiate a populace of particles
 then
  and for each particle p
                  define particle p
                   If p> pBest
        Then
       pBest -> p
    end for loop
 compute the gBest for all candidate particles
   for every particle p
 do
     calculate p ’s velocity
    Update p ’s position
   end_for
  while (stopping condition)
}
```

For TSP solutions on vertices ,As for each member of iterations there is 3 options :-

To follow and continue on its path.

To go back to its best solution in the past.

To go towards the global best solution.[30]

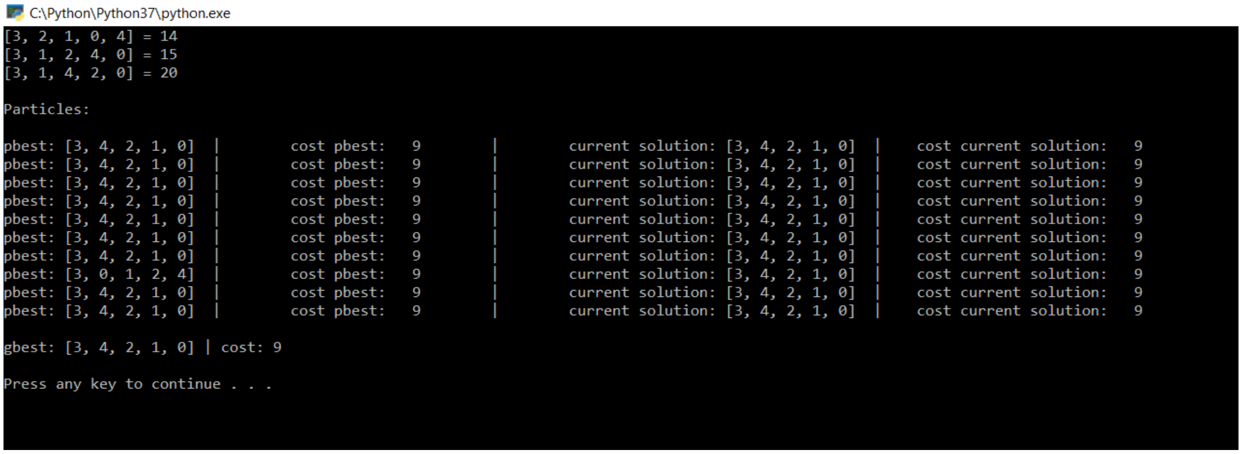
d)Complexity

1.Big-O :For the TSP algorithm using the PSO method it takes O(p*n^2) where p is the population size and n is the initial population and as the p is constant so its is O(N^2).

2. Big-Θ: Θ (p*n*log(n))

3.Big-Ω : Ω (p*n*log(n))

e) Output as in files:



```
[3, 2, 1, 0, 4] = 14
[3, 1, 2, 4, 0] = 15
[3, 1, 4, 2, 0] = 20

Particles:

pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 0, 1, 2, 4]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9
pbest: [3, 4, 2, 1, 0]  |       cost pbest:   9       |       current solution: [3, 4, 2, 1, 0]  |    cost current solution:   9

gbest: [3, 4, 2, 1, 0] | cost: 9

Press any key to continue . . .
```

G) Branch and bound optimization method:

1) Theoretical Foundation:

Branch and Bound algorithm (B&B) which is one of the methods for discovering ideal solutions for solving optimization issues specifically in the problems of discrete and combinatorial optimization ones that was first applied by A. H. Land and A. G. Doig. The branch is checked against upper and lower bounds on the optimal solution before counting of candidate solutions by methods for state space search where arrangement of up-and-comer solutions framed a rooted and all candidate solutions but not all nodes get expanded (children generation)but there is a criterion of carefully selected node that to expand just like the BFS and when finding the optimal solution.

It depends on the effective estimation of the lower and upper bounds of the branches in the inquiry space. If not found, the algorithm goes to an exhaustive hunt [31].

Considering an optimization problem let it B=(X,F) where X is a cohort of the solutions to the problem and F is the target value function. We want to find the optimal solution by building the sub problems of the search tree. Also we store the candidate solutions F(**x**) of last better objective value.

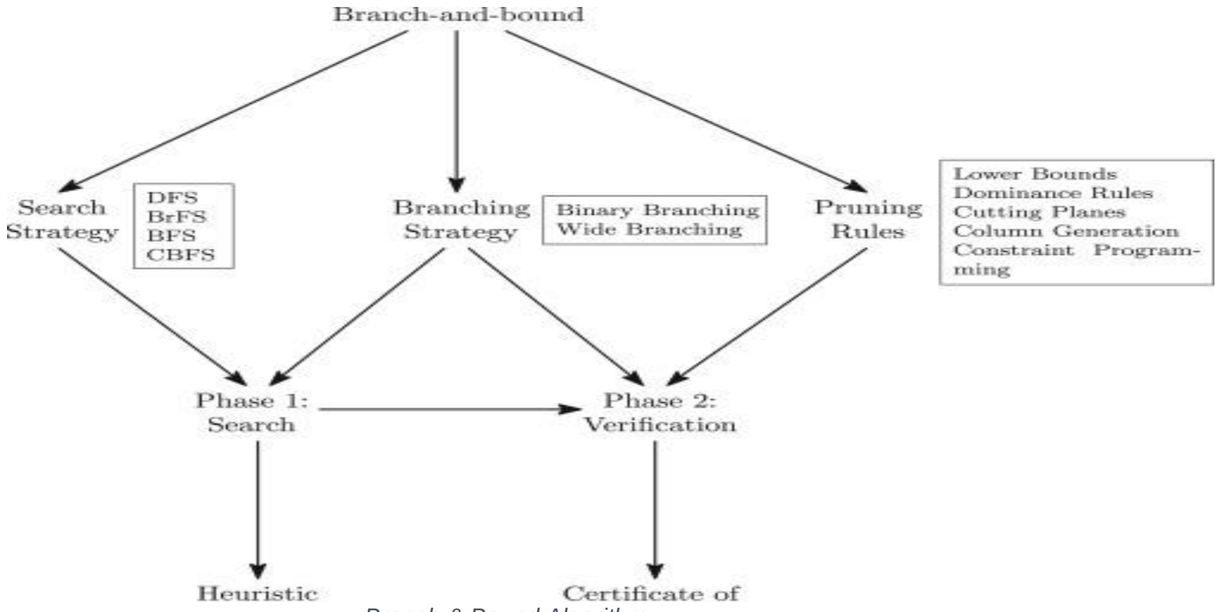
2) How it works:

There is 2 phases in the Branch and Bound algorithm the first is search where the algorithm has not observed the optimal solution whereas the second is the validation in which we can’t assure that the holding solution is the optimal as there is sub problems that hasn’t been explored yet. As we have said the B&B algorithm has 3 components .However, in the branch and bound mechanism consists of 3 components:-

Search Strategy: In which order did the sub problems explored its in the search phase so after determining the pruning rules the search mechanism see all the sub problems if the optimal solution has appeared.

Branching Strategy: How the space of the solution is divided in the tree to produce the new sub problems) It can be on both the search and validation phase, where it can guide the whole algorithm to the optimality.

Pruning Rules: This prevents the discovery of tree regions considered during the verification phase.



Branch & Bound Algorithm

3) When to use:

B&B functions are used to be combined with current values of solutions to help the solution space search algorithm and to solve the NP combinatorial problems [33].

A B&B algorithm for a given particular problem searches the solution space to find the solution. As the number of candidate solutions although, an explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions. The use of bounds for the optimized function combines with the value of the current best solution enables the algorithm to search parts of the solution space as the normal search will take an exponential time.

4) Global convergence to the optimal solution:

After conducting the search phase the right branching strategy limits the branching decisions that mimics the unwanted work to assure the optimality. Off the case that the algorithm ends reaching the optimality, the incumbent solution will return as a heuristic solution which could be sufficient for a bunch of problems.

Also to find an optimal solution has an impact on the size of the tree as for the time necessary to confirm optimality as there is no nodes needed to explored larger than the one for the optimum value explored[34].

5) TSP using the branch and bound method

a)Problem Domain:

Although branch-and-bound appeared in the field of integer programming, but it can also used in solving many combinatorial optimization problems[35].

It may also be considered related to numerous kinds of problems as they depend on the total set of solutions that can be divided into smaller set of problems which could be combined until finding the best solution (optimal).Used also for solving the NP-hard problems such as:-

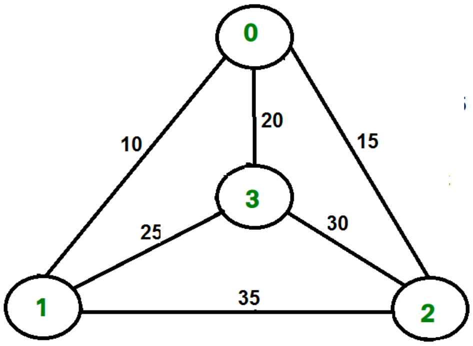
TSP, Nearest Neighbor Search, Integer Programming,,etc.

b) Limitations:

High time-consuming: As the number of nodes could be very huge in the tree, Also the developing branches will lead to a large number of offsprings so the tree may grow exponentially without enhancing the best solution.

c) Explanation of code:

Given some cities and the distance between 2 consecutive cities , our goal is to find the least costly tour in the condition that we visit all the cities exactly one and return to the starting position. As for the following graph of TSP tour.



- Bestpath array stores the best path founded by the salesman
- MinWeight variable stores the final least weight of the shortest tour.
- pathCopying()is a method that copies the solution to the best solution temporary
- FMinimum() is a method that finds the minimum edge cost at the final vertex.
- SMinimum()is a method that finds the second least cost of the edge at the final vertex.

- TSP() takes the lower limit of the root and store the cost of the path, current level in which in the space of exploration, store the best path.
  - The base case when we reached the level and already covered all the nodes.
  - Here we checked if there is an edge from the last visited vertex in the path traversed to the first one.
  - We then updated the final result if we found a better result.
  - We built the tree of the searching space by the for loop.
  - We calculated the considered bound for level 2 than the other levels.
  - We keep updating the lower limit by adding the current bound with the current weight, that if the lower limit is still less than the minimum weight , we keep exploring the node more.

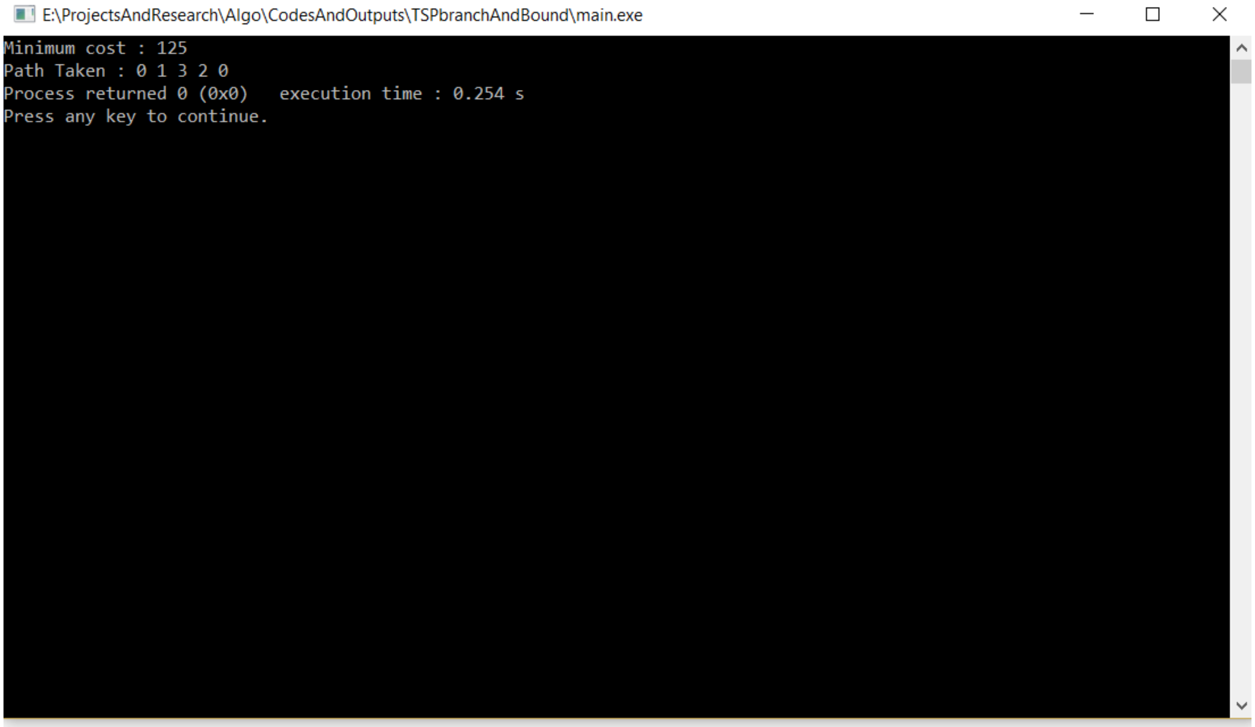
As discussed for a selected node that found in the traverse we want to calculate the candidate solution if we traverse down the node, If the bound on the best solution is costlier than the selected best (best calculated until now) we ignore the following subtree.

d) Complexity:

The time complexity of Branch and Bound algorithm will be similar as we calculated before the TSP Brute Force As in the least case we may cant traverse a node. Although in practice it goes well on the TSP problems. Also as the dependent factor here is on the branching strategy as it determines how many nodes to traverse.

1.Big-O : O(N!)).
2. Big-Θ: Θ(N!)
3.Big-Ω : Ω (N!)

e) Output as in the files:


E:\ProjectsAndResearch\Algo\CodesAndOutputs\TSPbranchAndBound\main.exe

```
Minimum cost : 125
Path Taken : 0 1 3 2 0
Process returned 0 (0x0)   execution time : 0.254 s
Press any key to continue.
```

IV) Comparison and results

- We successfully talked about the TSP problem by different approaches and techniques and I will conclude the results in the following table, we concluded that :-
  - Brute force complexity is same as the branch and bound.
  - Divide and conquer isnt much useful for the tsp.

Method or approach	**Big-O**	**Big-theta**	**Big-Omega**
Dynamic Programming	O($N^2 2^n$)	O($N^2 2^n$)	O($N^2 2^n$)
Brute Force	O ((N −1)!)	O ((N −1)!)	O ((N −1)!)
Greedy	O($N^2 * \log (N)$)	O($N^2 * \log (N)$)	O($N^2 * \log (N)$)
Genetic Algorithm	O(O*P*N^2)	O(O*P*N^2)	O(O*P*N^2)
Particular swarm inteligence	O(p*n^2)	O(p*n^2)	O(p*n^2)
Branch and bound	O(N!)).	O(N!)).	O(N!)).

Divide and conquer	-----	------	-----

V) REFERENCES

1. Rothlauf, F., *Optimization Methods*, in *Design of Modern Heuristics: Principles and Application*. 2011, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 45-102.
2. Talbi, E.-G. and G. El, *Metaheuristics: From Design to Implementation*. Metaheuristics: From Design to Implementation. Vol. 74. 2009.
3. Cormen, T.H., et al., *Introduction to Algorithms, Third Edition*. 2009: The MIT Press.
4. *CHAPTER 4 - Fundamentals of algorithms. 2009: p. 173 - 234.*
5. *David B. Wagner, "Dynamic Programming", THE MATHEMATICA JOURNAL, Miller Freeman Publications, 1995.*
6. *Muniswamy, V. V. (2010). Design and Analysis of Algorithms, I.K. International Publishing House Pvt. Limited.*
7. *Appligate, D.L., Bixby, R.E., Chavatal, V., Cook, W.J.: The Travelling Salesman Problem, A Computational Study. Princeton Univesity Press, Princeton (2006).*
8. *Abdulkarim, H. and I. F. Alshammari (2015). "Comparison of Algorithms for Solving Traveling Salesman Problem. International Journal of Engineering and Advanced Technology*.
9. *A.Kazakov,"Travelling Salesman Problem:Local Search and Divide and Conquer Working Together,"2009.*
10. *Chapter 2 - Computational Intelligence in Smart Grid Environment. Intelligent Data-Centric Systems, 2018: p. 23 - 59.*
11. *Shiroma, P.J., Theoretical Foundations of Genetic Algorithms, in Efficient Production Planning and Scheduling: An Integrated Approach with Genetic Algorithms and Simulation. 1996, Deutscher Universitätsverlag: Wiesbaden. p. 43-61.*
12. *Tomáš Kuthan and Jan Lánský. "Genetic Algorithms in Syllable-Based Text Compression". 2007. p. 26.*

13. *8 - Optimizing apparel production systems using genetic algorithms. Woodhead Publishing Series in Textiles, 2013: p. 153 - 169.*

14. *J.E. Baker (1987) Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette (ed.), Proceedings of the 2nd International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 14–21.*

15. *J.J. Grefenstette (1986) Optimization of control parameters for genetic algorithms. IEEE-SMC, SMC-16, 122–128.*

16.. *De Jong, K. A. (1990b). Introduction to the second special issue on genetic algorithms. Machine Learning, 5(4), 351–353.*

17. *Alajmi Ali, Wright, Jonathan(2014) Selecting the most efficient genetic algorithm sets in solving unconstrained building optimization problem. International Journal of Sustainable Built Environment, 18-26*

18.. *K. Helsgaun, "Effective implementation of the Lin-Kernighan traveling salesman heuristic," European Journal of Operational Research, vol. 126, no. 1, pp. 106–130, 2000.*

19. *C. R. Reeves, "Using Genetic Algorithms with Small Populations," Proceedings of the 5th International Conference on Genetic Algorithms, 1993.*

20.. *Ardalan, Z., Karimi, S., Poursabzi, O., & Naderi, B. (2015). A novel imperialist competitive algorithm for generalized traveling salesman problems. Applied Soft Computing, 26,*

21. *Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the 6th international symposium on micro machine and human science, pp 39–43*

22. *Ozcan E, Mohan CK (1998) Analysis of a simple particle swarm optimization system. In: Intelligent engineering systems through artificial neural networks, pp 253–258*

23. *Clerc M, Kennedy J (2002) The particle swarm-explosion, stability and convergence in a multi dimensional complex space. IEEE Trans Evolut Comput 6(2):58–73*

24. *1. Adewumi, A. O. & Arasomwan, A. M. (2015). Improved particle swarm optimizer with dynamically adjusted search space and velocity limits for global optimization. International Journal on Artificial Intelligence Tools, Vol. 24, No. 05,*

25. *Almufti, S., et al. (2019). "A comparative study of particle swarm optimization and genetic algorithm." 8: 40-45.*

26. *Kennedy J, Eberhart RC (1995) Particle swarm optimization? In: Proceedings of the IEEE international conference on neural networks, pp 1942–1948*

27. *Chen W, Zhang J, Lin Y, Chen N, Zhan Z, Chung H, Li Y, Shi Y (2013) Particle swarm optimization with an aging leader and challenger. IEEE Trans Evolut Comput 17(2):241–258*

28. *Shi, X.H.; Liang, Y.C.; Lee, H.P.; Lu, C. & Wang, Q.X. (2007). Particle swarm optimization based algorithms for TSP and generalized TSP, Information Processing Letters, Vol. 103, pp. 169-176*

29. *Elizabeth F. G. Goldbarg, Marco C. Goldbarg and Givanaldo R. de Souza (September 1st 2008). Particle Swarm Optimization Algorithm for the Traveling Salesman Problem, Traveling Salesman Problem, Federico Greco, IntechOpen,*

30. *Clausen, Jens. "Branch and bound algorithms-principles and examples." Department of Computer Science, University of Copenhagen (1999): 1-30.*

32. *Morrison, D. R., et al. (2016). "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning." Discrete Optimization 19: 79-102.*

33. *Clausen, J. (1999). Branch and Bound Algorithms - Principles and Examples. Copenhagen: University of Copenhagen.*

34. *D.R. Morrison, E.C. Sewell, S.H. JacobsonAn application of the branch, bound, and remember algorithm to a new simple assembly line balancing datasetEuropean J. Oper. Res. (2013)*

35. *Essien, A., et al. (2019). "APPLICATION OF BRANCH AND BOUND ALGORITHM IN A BOTTLING SYSTEM." 5: 72-84.*