International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600

http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646

Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

# A BACKTRACKING AND BRANCH & BOUND ALGORITHM USING KNAPSACK PROBLEM

**Dr.V.Selvi#1 Assistant Professor**
Department of Computer Science
Mother Teresa Women's University
Kodaikanal-624101.
1selvigiri.s@gmail.com

**G.Sadhana*2 M.phil Research scholar**
#Department of Computer Science
Mother Teresa Women's University,
Kodaikanal- 624101.
1selvigiri.s@gmail.com

**Abstract -** This paper describes what is termed as backtracking using maze problem and what is termed as branch & bound using Hamiltonian cycle. A backtracking algorithm is a recursive method of building up feasible solutions to a combinatorial optimization problem one step at a time. A backtracking algorithm is an exhaustive search, that is, all feasible solutions are considered and it will thus always find the optimal solution. It is a generalized of the ordinary maze problem to find a path from start from finish. One or more sequences of choices may lead to a solution. Many of the maze problem can be solved with backtracking. Branch and bound (BB, B&B, or BnB) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration. The algorithm explores branches of this tree, which represent subsets of the solution set. Using a Hamiltonian cycle a path which passes once and exactly once through every vertex of G (G can be digraph).

**Keywords--**Backtracking, branch&bound, maze, Hamiltonian, optimization.

## I. INTRODUCTION

A backtracking algorithm is a recursive method of building up feasible solutions to a combinatorial optimization problem one step at a time. A backtracking algorithm is an exhaustive search, that is, all feasible solutions are considered and it will thus always find the optimal solution. Pruning methods can be used to avoid considering some feasible solutions that are not optimal. To illustrate the basic principles of backtracking, we consider the Knapsack problem. Recall that a problem instance consists of a list of profits, P =[ p1,...,pn];

International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646
Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

a list of weights, W=[w1,...,wn]; and a capacity, M. These are all positive integers. It is required to find the maximum value of pixi subject to wixi ≤ M and xi ∈{ 0,1} for all i. An n-tuple[ x1,x2,...,xn] of 0's and 1's is a feasible solution ifwixi ≤ M.One naive way to solve this problem is to try all 2n possible n-tuples of 0's and 1's. We can build up an n-tuple one coordinate at a time by first choosing a value for x1, then choosing a value for x2, etc.

## II. PROPOSED APPROACH

### A. Backtracking Algorithm

A backtracking algorithm is an exhaustive search, that is, all feasible solutions are considered and it will thus always find the optimal solution. Backtracking provides a simple method for generating all possible n-tuples. After each n-tuple is generated it is checked for feasibility. If it is feasible, then its profit is compared to the current best solution found to that point. The current best solution is updated whenever a better feasible solution is found. We will denote by X =[ x1,...,xn] the current n-tuple being constructed, and Cur P will denote its

1.Algorithm:

```
Global X, Opt X, Opt P
If  l=n+1
Then
{
If  ∑n  i=1 wixi ≤M
Then
{
Cur P            ∑n  i=1 pixi
If cur P > Opt P
{
Opt P            Cur P
Opt X [x1,.............,xn]
Else
{
xl1
Knapsack1 (l+1)
Xl0
Knapsack1 (l+1)
}
}
}
}
```

Example:

2. Solving a Maze:

Given a maze, find a path from start to finish .At each intersection, you have to decide between three or fewer choices you don't have enough information to choose correctly. Each choice leads to another set of choices. One or more sequences of choices may (or may not) lead to a solution. Many types of maze problem can be solved with backtracking

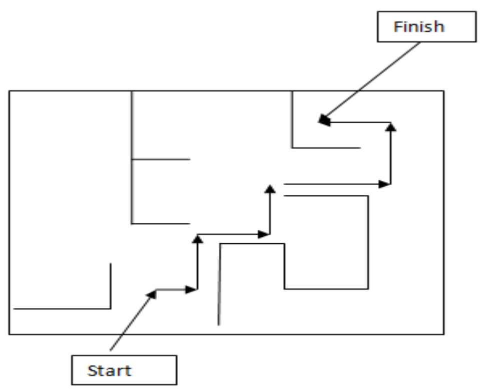



Fig1: Maze Problem

International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646
Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

- The bicycle lock problem:
- Consider a lock with N switches, each of which can be either 0 or 1.
- We know that the combination that opens the lock should at least N/2] 1's.
- Note: The total number of combination is 2N
- The solution space can be modelled by a tree

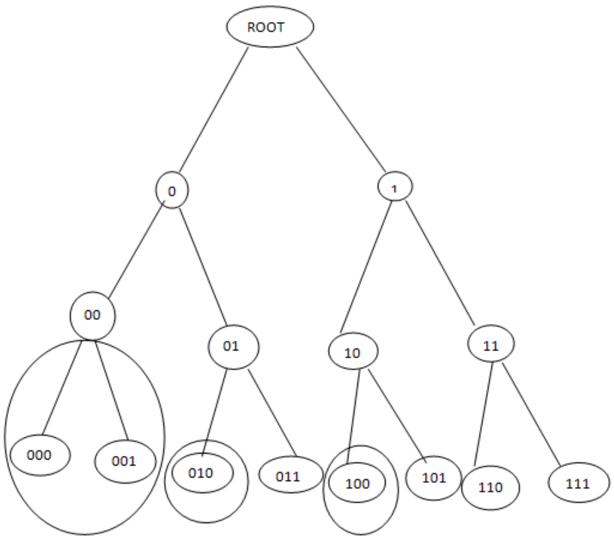



Fig 2: Rooting Tree

For some problems, the only way to solve is to check all possibilities.

- Backtracking is a systematic way to go through all the possible configurations of a search space.
- We assume our solution is a vector (a(1),a(2),
- We assume our solution is a vector (a(1),a(2), a(3), ..a(n)) where each element a(i) is selected from a finite ordered set S.

3. General Algorithm:

```
Procedure backtrack()
/* X is the solution vector */
Integer k;
Begin
K=1;
Compute sk; /* Compute the possible solution values fork=1 */
While k>0 do
While sk<>Φ do
X[k]=an element of sk;
Sk=Sk-{x[k]};
If B(x[1],........x[i],........x[k])=True
Then print the solution vector x;
Else begin
        K=k+1;
        Compute Sk;
End;
End while;
        K=k-1;
End while
End;
```

4. Recursive Solution:

Backtracking is easily implemented with recursion because The run-time stack takes care of keeping track of the choices that got us to a given point of the choices that got us to a given point. upon failure we can get to the previous choice simply by returning a failure code from the recursive call.

```
Procedure back_recursive (k)
Begin
        For each x[k] in Sk do
        If B (x[1],......,x[i],.........,x[k]=True
        Print the solution vector x;
Else begin
        Compute Sk;
```

International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600

http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646

Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

Back_recursive (k+1);
End if;
End for;
End

B. Branch and Bound Algorithm

Branch and bound (BB, B&B, or BnB) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. 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.[3]

Example

1. Hamiltonian Cycle:

Hamiltonian cycle (HC): is a cycle which passes once and exactly once through every vertex of G and returns to starting position[6]

Hamiltonian path: is a path which passes once and exactly once through every vertex of G (G can be digraph).

A graph is Hamiltonian if a Hamiltonian cycle (HC) exists

2. Hamiltonian Circuit:

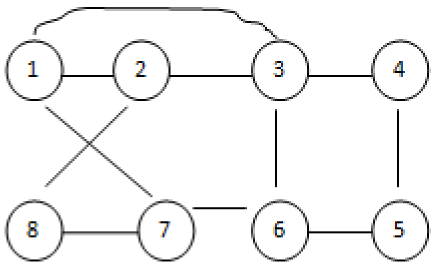



*Fig 3: Graph1*

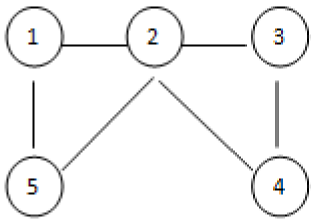



*Fig 4:Graph2*

- Graph G1 contain hamiltonian cycle and path are 1,2,8,7,6,5,3,1
- Graph G2contain no hamiltonian cycle.
- Here solution vector (x1,x2,...,xn) idefined so that xi represent the I visited vertex of proposed cycle.
- The algorithm is started by initializingadjacency matrix G[1:n,1:n,then setting x[2:n] to zero & x[1] to 1, then executingHamiltonian(2)

Algorithm
Algorithm Nextvalue(k)
{
repeat
{

http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646

Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

x[k] := (x [k] +1 mod (n +1)
//next vertex
If ( x[k] = 0) then return
If G[x[k-1], x[k] ≠ 0 ) then
{
//Is there an edge?
for j =1 to k-1 do if (s[j] = x[k]) then break;
//check distinctness
If ( j= k) then//if true then vertex is distinct
If (( k< n ) or (( k =n) and G [x[n] , x[1] ≠ 0))
then return

}
} until ( false) ;
}

3. Hamiltonian Algorithm

Hamiltonian path: is a path which passes once and exactly once through every vertex[5]

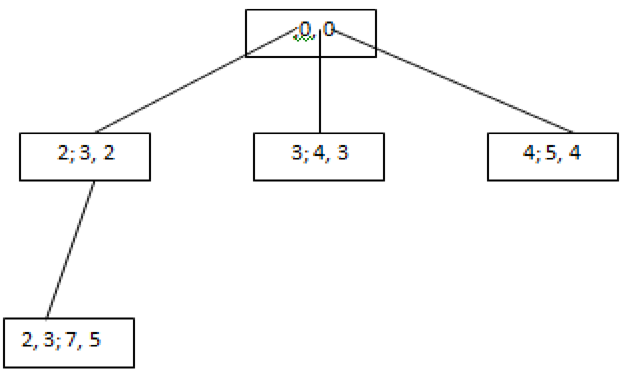
Algorithm Hamiltonian (k)
{
repeat
{
//generate values for x[k]
Nextvalue (k);
//assign a legal next value to x[k]
if ( x[k] = 0 ) then return
if (k = n) then write ( x[1:n]);
else
Hamiltonian(k + 1);
} until(false);
}

Knapsack Problem

Here instead of considering no. of items, we consider that we have n types of items & that proper no. of items of each type is available. This 0/1 knapsack problem

algorithmBoundKnapsack(T,W)
{
b=0
for i= 1 to n do
if(w
i
≤W) then
{
b=max(b, pi+ BoundKnapsack(I,W-wi))
}
return b
}

T=<T1,T2,T3>
w=<2,3,4>
p=<3,4,5>
W=5



International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
http://www.ijcsjournal.com Volume 5, Issue 1, No 21, 2017 ISSN: 2348-6600
Reference ID: IJCS-257 PAGE NO: 1641-1646
Alagappa University, Karaikudi, India 15th -16th February 2017
IT Skills Show & International Conference on Advancements in Computing Resources (SSICACR-2017)
http://aisdau.in/ssicacr ssicacr2017@gmail.com

For example,, if we once visit node(2,3; 7,5) then next time we do not visit node(3,2;7,5). The first node visited is (2;3,20 the next is (2,3;7,5). It can be seen that as each new node is visited the partial solution is also extended. After visiting theses two nodes the dead end comes as node(2,3;7,5)has no unvisited successor, since adding more items to the partial solution violates the knapsack capacity constraint we memorize it. This is optimal solution for our problem with maximum capacity 7 and T1 & T2 are include into the knapsack[6]

III Conclusion

Both Backtracking and Branch&bound algorithm try to find out the optimal solution. In both algorithm an optimal solution to the problem contains within it optimal solutions. We have shown how the Hamiltonian Cycle problem is equivalent to both solving a system. So In future we need to develop algorithm that trade-off between parallelism and expenses.

REFERENCES

[1] T.H.Cormen, C.E.Leiserson and R.L.Rivest, Introduction to algorithms, I ITPress, Cambridge MA,1996.
[2] Levitin, Anany. The Design and Analysis of Algorithms. New Jersey: Pearson Education Inc., 2003.
[3] Different Approaches to Solve the 0/1 Knapsack Problem. Maya Hristakeva, DiptiShrestha; Simpson Colleges
[4] S. Dasgupta, C. H. Papadimitriou and U. V. Vazirani.Algorithms
[5]https://en.wikipedia.org/wiki/Hamiltonian_path_problem
[6]https://en.wikipedia.org/wiki/Maze_solving_algorithm