An Adaptive Algorithm For Recursion Based Games

Gautam Das
Roll No: 01010107

Siva Gurumurthy
Roll No: 01010125

Sumit Suman
Roll No:01010128

1. AIM

To design a strategy for playing a *N-Polygonal* game where the person who picks up the last coin loses .

2. INTRODUCTION

A learning based environment was created under the sustained conditions to maintain high points and secure wins . Heavy recursion was required as the number of moves possible at initial increases exponentialy with the number of coins . A Coin is placed in a sub-block of a polygon , we designed an hetrogene of both dynamic and adaptive algorithm to play square and triangle game. Algorithm was designed for playing n-polygonal game . just the number of coins and the shape of the polygon has to be fed into the system and rest will be taken care of. In our impementation part ,we have started from scratch and then sequentially incremented our functionalities with every phase . All algorithms of all the phases are briefly described in this report.

3. PHASE: 1

We were using a simple algorithm for reducing the opposition to odd number of moves available ,that will make the opposition pick the last coin. file then used is now cs20402_random.cpp ,we still use some functions from it.

3.1 Algorithm

when one move is played by opposition , compute all possible moves available at that moment .

next move of the game is calculated by calculating the moves that would make the possible movs to odd number.

3.2 Advantage

The bast advantage of the first round algorithm is that it is very fast.

3.3 Drawbacks

On the other, it has very few number of wins.

Thr game conducted is during the algorithm course : cs204.
Department of Computer Science and Engineering
Indian Institute of Technology
Guwahati, Guwahati : 781039
.

4. PHASE 2

Idea of log files were introduced into the preexisting assignment 1 file , which makes it a bit of learning (not extensive) . It would play the game from the log file if it were present otherwise it follows the assignment 1 technique;

4.1 Algorithm

Same game as previous one with some logging of already won games , so that when it wins or loses wih opposition , it makes logs of it and then moves the same game if same context is available.

4.2 Advantage

In this case, the game was fast , others win can be used for its game as learning strategy.

4.3 Disadvantage

Disadvantages are few wins, as large number of games would be required to play to increase learning.

5. PHASE 3

More improvement on the pre-existing files , apart from this recursion was introduced (back tracking principle) , its idea is to check for all possible moves at particular instant , if it can win at least one of those moves, it will move that move. Hence, anything that comes out of the recursion will be a sure solution, memoization of all possible moves at particular instant, a bit of learning that without much improvement from former, a bit of greedy choice.

5.1 Recursive Function

```
recurse(initial_context)
{
  calculate all_possible_movs
  for( i = 1 to number_of( all_pos_movs ))
  {
    mov[i] + initial_context;
    present_moves;
    recurse(present_context);
    if(child[i] "loses")
      continue with other children
    else
      return "i lose";
  }
  write context;
  return "i win "
}
```

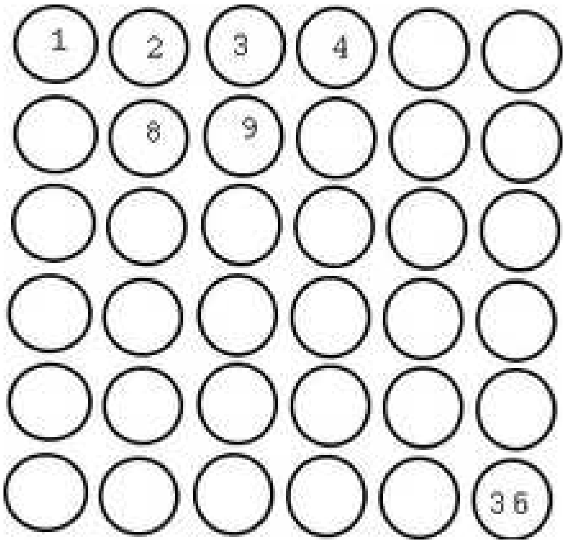

Figure 1: Square game with 36 coins

the context which comes out of the recursion will be a sure win one ; if there is no soln then , it will be a worst time case.

$$timecomplexity = number\_of\_possible\_moves \times$$

$$T(number\_of\_possible\_after\_the\_move)$$

Since the number of possibe moves increses exponentialy with number of coins, hence time complexity is exponential.

5.2 Advantage :

This approach enables large number of wins.

5.3 Disadvantage :

In this method, very heavy recursion was used. So there were some timing problems leading to time outs, point management was not satisfactory , as there were some other teams who scored less number of wins and still bettered score than us.

6. PHASE 4:

Further improvement in reducing time by splitting the recursion into two portions one would be activated if the numbers of coins goes less inside the recursion , other would use hash table policies for improving the recursion time, both involves memoization principles.the recursion time reduced by 5 times approximately ,apart from this extensive greedy choice algorithm is used , every time when the game is in losing mode , it becomes greedy and picks then maximum points , some logs were used , all the files that were pre existing are used because every one corresponds to particular strategy or functions. Since the code was generated in a generic way , triangle to square game was converted in no time .

```
Recursion
  {
  if (coins_left > 14)
    {
       use memoization and make the flag = 0 for
       which the moves is played and other moves
       that would not be there for same number of
       coins missing
    }
    else
    {
      use hash table and find all the moves that
      correspond to
      existing coins;
    }
    same as above ;
  }
  if recursion fails()
  {
    Move the game that earns u maximum points;
  }
```

6.1 Advantages

The most important think is that the game tried to have large number of wins, maximum points gained, effective use of time, winning games in good margins.

6.2 Disadvantages

not completely dynamic , though bit of learning was used , no learning was seen in the game played.

7. FINAL PHASE

Highly dynamic environment is fed into system , its an extensive of the previous files and highly greedy , it does maximum optimization and there cant be a better than solution at that context , highly learning capabilities , if a game is lost , next time it plays a different game with points optimization, after repeated executions against non dynamic opposition gives a yielded points , some times it wins after losing in successive times ,but never gives in the name of learning , it will make a different move only if there is same or more points than present. other techniques that are inherited from previous assignments are recursion , memoization, hashing techniques, greedychoice, back tracking.

7.1 Algorithm Used

```
computemoves();
sort_accordance_to_scores();
store_it_in hashtable;

learning()
{
  lost =open("lost.log");

if (present_context = lost)
  change_move = true;
if (present_context = won)
  play_from_memoized_file;
}
driver();
{
  present_context;
  soln = recurse(present_context);
  if(soln == null)
    {
      if ( not change_move)
```

```
      pick the first vals of hash table
      // that will give maximum score;
      if(change_move)
 pick the succesive val from
         previous context proceed;
    }
    if(game_lost)
    write in "lost.log"
    if(game_won)
      if ( player 1 wns)
         write in win1.log
      if (Player 2 wins)
         write in "win2.log"
}
{
recurse()
  {
    same_As above
    computemoves() using point strategy;
  }
}
```

7.2 Advantages

The the advantages of the algoriithm that we have used during the fifth round uses advanced learning, try to maximize points, number of wins would be evident after the games are played.

7.3 Disadvantages

In this final game, we have tried to remove all the flaws present in the game. There is hardly any disadvantages.

8. GENERALLITY

Since every assignment was a scope of imrovement for the next, very last one is the game that is endowed with all the features , Generic way of coding was followed to generate the number of moves . Suppose for a n-polygon containing m number of coins and hence following function was made

```
compute_moves(starting_point , ending_point, difference ,
is_increasing ,increases_by)
{
  last_diff = difference;

  if(is_increasing)
  {
    move[i] = move[i-1] + last_diff;
    last_diff = last_diff + increases_by;
  }
  else
  {
    move[i] = move[i-1] + last_diff;
  }
  return move;
}
```

this would return all possible moves between the limits which we specify and hence as soon as it gets computed , it stores the moves in a hash table to enable memoization techniques . so for a n - polygon, when m coins are evenly distributed them ,

$F : (R \times R) \longrightarrow R$

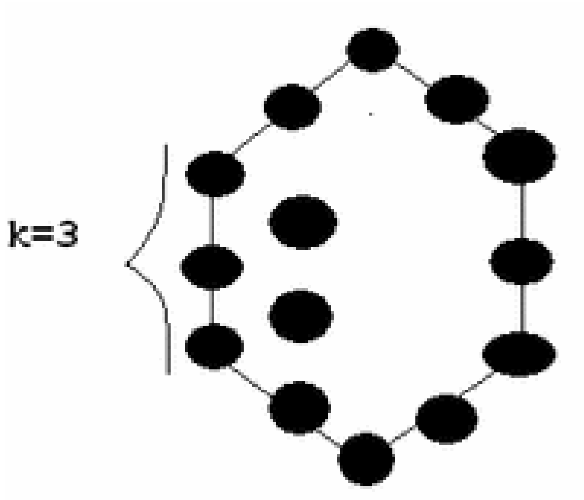

Figure 2: Hexagonal Game Structure

$F(m, n) = k$; where k will be the minimum number of coins that are distributed along an edge of the polyon, so computation of k will enable us to compute all the pairs of starting points and end points and differnece between them .

F is a function that is purely geometry dependent , for example for a hexagon

$n = 6$

$m = 19$ $numberofcoins$

suppose k be an integer which satisfies all the requirements

$$(k) + (k + 1) + (k + 2) + (k + 1) + (k) = m$$

$$5 * k + 6 = m$$

f = 5k + 6 - m , any k which is an integer that satisfies f=0 will be a solution to the equation. assuming the distribution is even .

on computation k = 3 , since k is integer our assumption of the geomety is hence proved . geometry is hence appears like figure 2.

9. MEMOIZATION

In the last section we have seen how a possible geometry can be suggested, if the solution that comes after the computation of the function is valid then its limits that will arise out of the geometry will help us to find all possible moves for that geometry . Time complexity for determining all possible moves t(n,m) is dependent upon geometry dependent, WLG it can be assumed that the limits between any two points will be maximum for a triangle (n = 3) (infact our assumption can be proved right as same number of coins are distributed in n - polygon where n ¿ 3) now, coins will be distributed in this form

$1 + 2 + 3 + 4 + ... + k = m$;—- (1)

$k(k+1)/2 = m;$

$k^2 + k = 2m;$

$k^2 + k - 2m = 0;$

$k = (-1 + \sqrt{(1+8m)})/2$

$k < \sqrt{(m)}$—-(2)

Since k is the maximum number of coins that can be accomodated in an edge , so in that row of k coins number of possible moves would be

k(single coin moves)+ k-1(double coin moves) + k-2(triple coins) + ... 1(k coin move) so that would be $k + k - 1 + k - 2 + ... + 1$ number of moves ib that k th row . from (1) it equals m ; since there are three edges similar to it , 3m-3 would be maximum number of moves as three corners have intersection between edges , now one but last row will have m-k number of moves , that accounts for 3(m-k)-3 total moves . on recursing the same .

Total number $= 3m-3+3(m-k)-3+3(m-k-(k-1))-3....$

$= 3m(k-1) - 3(k-1) - (k + k + (k-1) + k + (k-1) + k + (k-2) + (k-3)....)$

$< 3mk$

$T(m) = O(3mk) and from (2)$

$T(m) = O(3m^(3/2))$

since the time required for the computation is reasonably large , at a competitive environment every instant we cannot afford to spend so much time for computing the possible moves , but we can observe the pattern that the moves are repeated , so we take advantage for computing the moves only one time for a game , so all the possible moves are stored in a hash table and whenever particular amount of coin is removed from the board , its corresponding hashtable value is called and the move is erased to indicate that no more moves available that are pertaining to the removed coins . Experimentally we found that the time that changed was from 90 secs to 15 secs , its a 600% reduction in time , so recursion could be further advanced by the memoization technique.

10. DYNAMIC ENVIRONMENT

Since the game was played between equally weighted players , the scores was very much similar and repetitive games yielded the same moves resulting in same number of points . To challenge this ,dynamic environment was fed into the system .

10.1 How It Works

A reputed file for learning is created on playing the very first game . it registers all the loses it has experienced and whence the same situation occurs , it refrains from moving the same move which has caused a defeat previously . Apart from this when it is checking for different move it also refrains moving the move which would earn a lesser points . so it was clearly evident from the repeated games that everytime there was a increase in points . infact , in ideal cases it ended up beating the opposition to whom it has lost initially .Algorithm for this is already explained . when a graph was plotted against the number of games played and the scores registered , there was a drastic improvement from first few games and there after the difference between started to decrease ,so after some time it remains constant .Reason for this since dynamicity is restricted by the point system , so when it loses , it looks for a different move that would earn equal or more points than its original one , since the number of moves possible at that particular context is restricted with limited numer of moves of same points , so once it loses to all the moves then it plays same move for this stage and at next stage it looks for a different move . So it is clear that the scores would decrease with games and finally coming to a stanstill. graph apears like following :

11. FUTURE WORK

Any such recursion which proceeds linearly , can be further optimized to recurse all the contexts parallely using peer to peer computing A part of recursion can be distributed to some other processors . whenever a solution is reached in one of the processor , other threads are dissmissed , by this way the time complexity would be reduced into linear time .

12. CONCLUSION

During the game conducted during the course, we came to conclusion that the algorithm that we developed is generalised to *N-Polygon*. The algorithm adapts according to the game. The algorithm used is a recursive algorithm that uses memoization, dynamic programming, backtracking. Hence a succesful adaptive based algorithm was designed and implemented without any discrepencies.

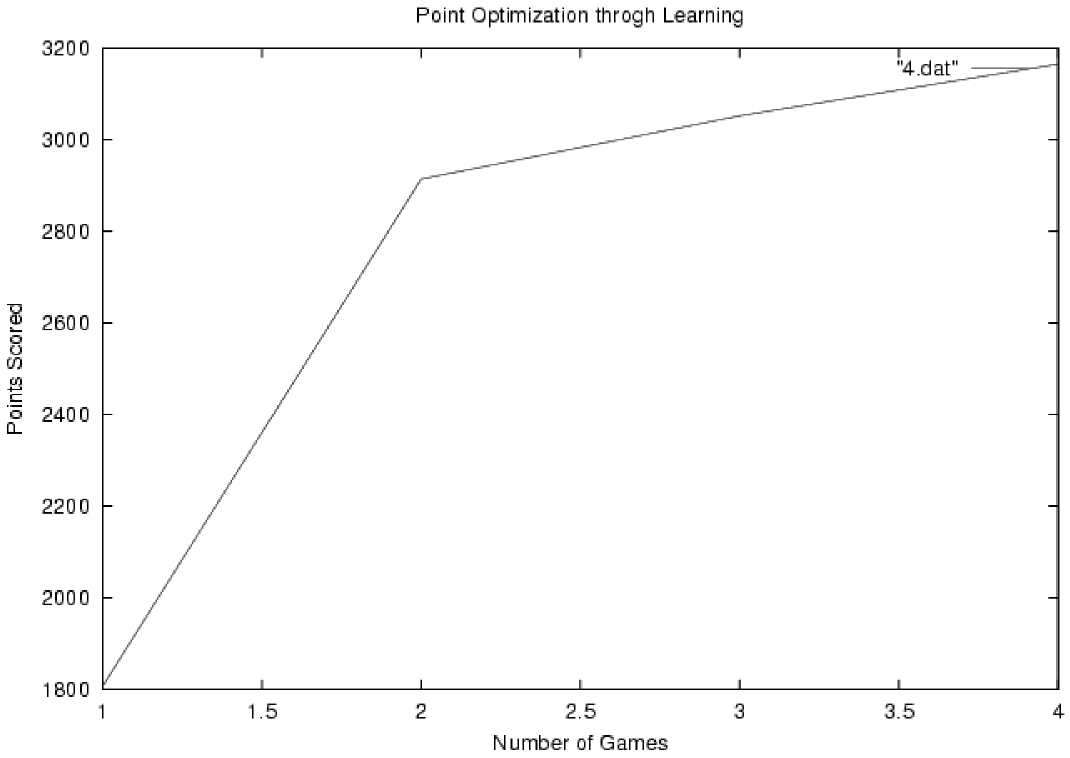

Figure 3: Example 1

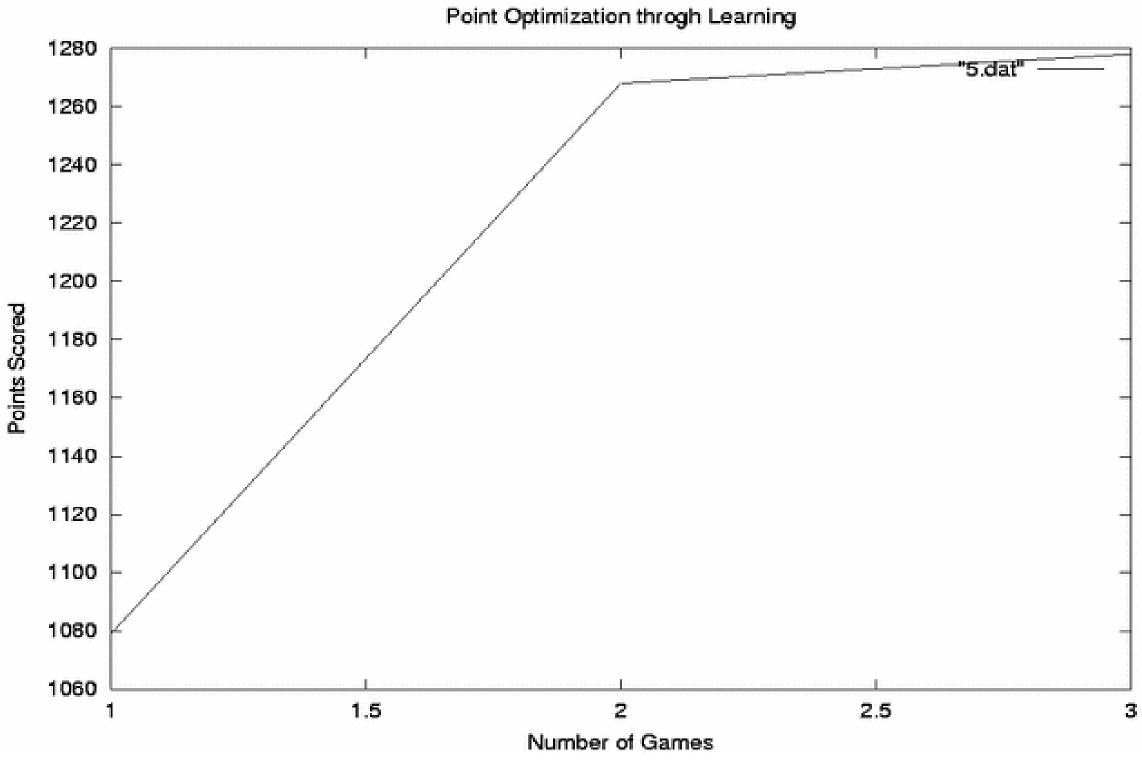

Figure 4: Example 2