CS152-Abstractions and Paradigms in Programming

Project Report

Vijaykrishna G, 170050090

Parikshit Bansal, 170050040

Rahul Bhardwaj, 170050012

A) Project 1: Othello Game-Playing AI

Basic Idea:

Our game-playing AI uses the Minimax Algorithm upto depths of 5-6 to determine the next best move given a board configuration. Additionally, Alpha-Beta Pruning has been used to further increase the efficiency of the algorithm. Different positions have been given different weightages based on heuristic estimates.

Input and Output:

The input and output is completely in GUI. We have used the racket/gui library for implementing gui and necessary input/output by clicks and keystrokes.

Additional Design Optimizations:

1) **Higher Order Functions:** Examples include: (convert i j dx dy color board), (allowed? i j dx dy color board). This function is used to convert all the to-be-captured pieces that are along the (dx dy) direction. These higher-order functions prove very useful in functions where we have to map over 8 directions.
2) **Usage of Macros:** This reduced a lot of code in the functions (allowed? I j dx dy color board) to traverse along the direction until a piece is met using while. We have used my-for macros for traversing through the whole board.

---

3) **Use of Debugger:** We used the Racket Debugger for debugging at various places.

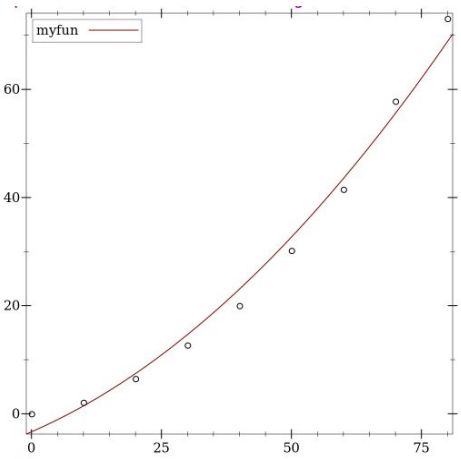
Limitations and Bugs:

1) The AI seems to be a bit weak at the start but plays well during middle and end game suggesting that there are some standard openings just like chess which the computer doesn't know.
2) For depth more than 6, it takes too much of time suggesting that there may be more advanced methods to further increase the depths.

B) Project 2: Function Fitter using multi-feature linear regression:

Basic Idea: We have used techniques of linear regression to fit an appropriate function given a basis of functions {f1, f2,...}.

Input and Output: We have 3 data sets data1, data2 and data3. Input format: (time (fd data2 basis2)) where basis2 is a quadratic basis (basis is also given by user).




```
#(-3.3308544256023644 0.4173072403613892
0.006061131206442443)
#<procedure:myfun>
cpu time: 443 real time: 444 gc time: 121
```

Additional Features/Optimizations:

- **Feature Scaling:** When the x and y of the data set are of different orders of magnitude normal regression leads to practical difficulties like very slow convergence. To increase the rate, we have used feature scaling on the initial data set to make the x and y of the same order.
- **Regularization:** When the number of features are high (a quartic fit for example), we have used regularization in order to reduce overfitting.
- **Display of error messages:** We keep track of the convergence/divergence dynamically so that we can print out “Slow Convergence” or “Divergence” and can visualize it by varying the value of alpha manually.

Limitations :

- For very large data sets, it takes a lot of time to converge, therefore we have limited our data sets to around 10 to converge within a second.
- We have to adjust alpha manually depending on the error messages, unlike advanced optimization algorithms like fminunc found in Octave.

Project 3: Sudoku Solver/Generator:

Basic Idea: We are solving and generating Sudoku using the standard backtracking algorithms also used in Prolog which we have implemented in Racket. We have created a GUI game for the same.

Input and Output: The input for the game is the difficulty level, which generates an appropriate Sudoku problem and interacts with the user giving hints and error messages for invalid moves.

Additional features and optimisations:

- Instead of checking each number from 1 to 9, we have generated a list of possible numbers, which reduces its time exponentially.
- While solving the board we have immediately exited after encountering the second solution rendering the problem invalid.

Limitations: The problem has been represented as a list of lists instead of vectors which increases its time by n^2.