Jeremie Pouly and Justin Fox
4/10/05

Project Proposal
16.412

An Empirical Investigation of Mutation Parameters and Their Effects on Evolutionary Convergence of a Chess Evaluation Function

Motivation

The intriguing and strategically profound game of chess has been a favorite benchmark for artificial intelligence enthusiasts almost since the inception of the field. One of the founding fathers of computer science, Alan Turing, in 1945 was responsible for first proposing that a computer might be able to play chess. This great visionary is also the man credited with implementing the first chess-playing program just five years later. In 1957, the first full-fledged chess-playing algorithm was implemented here at MIT by Alex Bernstein on an IBM 704 computer. It required 8 minutes to complete a 4-ply search.

Even in those early golden years of our field, it was recognized that the game of chess presented an exceptionally poignant demonstration of computing capabilities. Chess had long been heralded as the "thinking man's" game, and what better way to prove that a computer could "think" than by defeating a human player. Strategy, tactics, and cognition all seemed to be required of an intelligent chess player. In addition, early AI programmers likely recognized that chess actually provided a relatively simple problem to solve in comparison to the public image benefits that could be gained through its solution. Chess is a deterministic, perfect information game with no hidden states and no randomness to further increase the size of the search space. It was quickly recognized that brute force techniques like the mini-max and alpha-beta searches could, with enough computing power behind them, eventually overtake most amateur players and even begin to encroach upon the higher level players. With the advent of chess-specialized processors, incredible amounts of parallelism, unprecedented assistance from former chess champions, and the immense funding power of IBM, the behemoth Deep Blue was finally able to defeat reigning world champion Gary Kasparov in a highly-publicized, ridiculously over-interpreted exhibition match.

Having logged this single data point, the artificial intelligence community sighed contentedly, patted itself on the back, and seemingly decided that chess was a solved problem. Publications regarding chess have declined steadily in recent years, and very little research is still focused on ACTUALLY creating a computer that could learn to play chess. Of course, if you have a chess master instruct the computer in the best way to beat a particular opponent and if you throw enough computing power at a fallible human, eventually you will get lucky. But is chess really solved? More importantly to the project at hand, should we cease to use chess as a test-bed for artificial intelligence

algorithms just because Kasparov lost one match? (or rather because IBM paid him to throw the match? You will never convince me otherwise by the way! ☺)

We think not. The original reasons for studying chess still remain. Chess is still a relatively simple model of a deterministic, perfect information environment. Many currently active fronts of research including Bayesian inference, cognitive decision-making, and, our particular topic of interest, evolutionary algorithms can readily be applied to creating better chess-playing algorithms and can thus be easily benchmarked and powerfully demonstrated. This is the motivation for our current project. We hope to remind people of the golden days of artificial intelligence, when anything was possible, progress was rapid, and computer science could capture the public's imagination. After all, when Turing proposed his famous test, putting a man on the moon was also just a dream.

Project Objectives

1. Implement a chess-playing program which can be played human vs. human, computer vs. human, and computer vs. computer.
2. Re-implement the chess evaluation function evolution algorithm with population dynamics published by [1].
3. Conduct a parametric study of the mutation parameters used by [1] in an attempt to discover the dependency of the evolution's convergence on these parameters.
4. Suggest improvements to the mutation parameters used in [1] to make that algorithm more efficient and/or robust.

Technical Introduction

The focus of our work will primarily be the re-implementation of the evolutionary algorithm for evolving chess evaluation functions using population dynamics proposed and demonstrated by [1]. This algorithm first proceeds by defining a relatively simple evaluation function for a computer chess player given by a weighted combination of seven factors:

$$Evaluation = \sum_{y=0}^{6} W[y](N[y]_{white} - N[y]_{black})$$

where: N[6] = { N° pawns, N° knights, N° bishops, N° rooks, N° queens, N° kings, N° legal moves}

W[6] = { $weight_{pawn}$, $weight_{knights}$, $weight_{bishop}$, $weight_{rook}$, $weight_{queen}$, $weight_{king}$, $weight_{legal\ move}$ }

The parameter to be evolved is, of course, the weight vector **W**.

The original algorithm then creates an initial population of 50 alpha-beta chess-players each with the above evaluation function and its own random **W** vector. The weights are initially uniformly selected from the range [0,12]. The evolution then begins by allowing chess players to compete against one another in a particular fashion which ensures that stronger players are allowed to play more often than weak ones. Each match consists of two games with players taking turns as white or black. More games are not required since the algorithms are entirely deterministic and the outcome will therefore never change. After each match, if there is a clear winner, the loser is removed from the population. In its place, a mutated copy of the winner will be created. The winner may also be mutated in place. Mutations take place by adding or subtracting a scaled number onto each element of a population member's weight vector. Thus:

$$V_{(y)} = V_{(y)} + \left((RND(0..1) - 0.5) \times R \times \sigma_{(y)}\right) \qquad \forall y \in v$$

Player wins both games: Expel loser, duplicate winner and mutate one copy by R = 0 and the other copy by R = 2.

Player wins one game and draws the other: Expel loser, duplicate winner and mutate one copy by R = .2 and the other copy R = 1.

Players draw: Both players are retained and mutated by R = .5.

The astute reader will immediately note that the R values above seem rather ad hoc, and indeed Kendall and Whitwell note that the R values were "selected based on initial testing" and not by any theoretical or rigorous means. [1] In addition, the use of the standard deviation to control the rate of mutation while an interesting and possibly useful idea was simply proposed without further evidence to support its use. It will therefore be our purpose in this project to empirically discover evidence for or against the mutation parameter choices chosen by Kendall and Whitwell. The evolutionary algorithm community as a whole will benefit from this study in that we will be providing important evidence depicting how stable and robust an evolutionary algorithm is to changes in these mutation parameters. Is the selection of such parameters critical for stability and convergence of this type of algorithm, or do they merely affect rates of convergence in a minor or insignificant manner? Is Kendall and Whitwell's suggestion of using the standard deviation of the population as a metric for convergence a valid and necessary one, or are there other choices that produce similar results? Perhaps in answering these questions we may gain insight into an even better method of selecting such mutation parameters, or else discover that the parameters are inconsequential and a programmer need not waste time and energy carefully selecting them.

Previous Work

In choosing chess as our test-bed, we have the benefit of five decades of Herculean research upon which to build. We perused a number of papers during our background research phase and include references for a few of the more interesting of these. [10][11][12]