Jeremie Pouly and Justin Fox Project Proposal
4/10/05 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:

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

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]

In order to be able to understand Kendall and Whitwell's algorithm, we first had to understand the alpha-beta search algorithm upon which they based their arguments. The basics of this method are covered in, for example, [2], [3], and [4]. Since we have already demonstrated our in-depth understanding of this algorithm through our advanced lecture and demonstration and in order to not make this proposal more than 10 pages long, we will assume that the reader is familiar with this algorithm and the significant advancements which can be made to it.

In choosing to study the effects of mutation parameters on evolution convergence, we first needed to research the literature and see what studies, if any, had been conducted along these same lines. The number of papers on general evolutionary algorithms is astounding [5][7][8], it having become a veritable buzzword in the late nineties. However, most of these papers focus on solitary attempts to create an evolutionary algorithm for a particular field. Many fewer of the papers in the literature are actually detailed theoretical analyses of how an evolutionary algorithm should be created [6][9] and practically none provide detailed evidence as to why they chose their mutation parameters as they did. The reason for this gaping lack can probably be best summed up by a passage from [5]: "Probably the tuning of the GA [Genetic Algorithm] parameters are likely to accelerate the convergence [of the evolution]. Unfortunately, the tuning is rather difficult, since each GA run requires excessive computational time." It seems that authors have been generally much too concerned with turning out a paper containing a neat application in as short a time as possible and much less eager to invest the admittedly exorbitant amount of time and computational resources required to investigate this question. After this rather dismaying survey of the literature, we decided that our investigation would therefore be quite beneficial to the field though we too are daunted by the computational time requirements requisite of this ambitious undertaking.

Outline of the Proposed Technical Approach

We will be performing all of our implementation work in the Visual Basic language. This platform exhibits the benefits of simple to use and design graphical user interfaces as well as loose restrictions on programming style (implicit memory allocation, generally low overhead programming required, etc.). Perhaps even more importantly, our combined comfort level in Visual Basic is almost certainly higher than for any other applicable graphical language. Considering the ambitious amount of implementation we plan to undertake, selecting a language that is both familiar and simple to use could easily mean the difference between a successful project and a failed one.

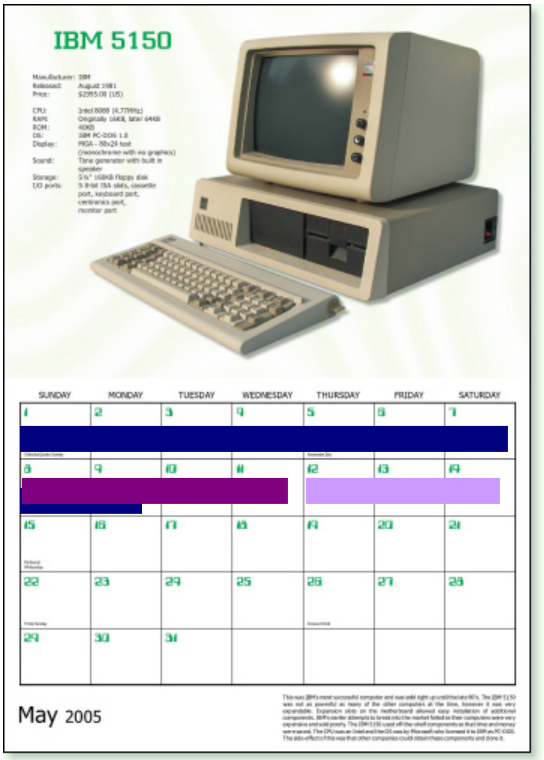
Our general plan is to divide the project into two basic phases. (Believe it or not, this was our plan even before the professor mentioned practicing a two-spiral design method.) The first spiral has been in progress almost from the very first day the project was explained to us. Its primary objective was to create an efficient alpha-beta checkers-playing algorithm and then to apply an evolutionary method similar to Kendall and Whitwell's to the evaluation function in checkers. This first phase should allow us to obtain a strong background in the general methodology associated with game programming, including the graphics required, alpha-beta search techniques and efficiency improvements to that method (transposition tables, quiescent search, move-

ordering, etc.), and the basic implementation details behind the evolutionary algorithm under consideration. Checkers is a relatively simple game in comparison to chess, making our programming tasks less prone to bugs and much more efficient to test. Indeed, allowing two computer opponents which search to 3-ply to play a checkers game requires only about 4 seconds whereas our best estimates so far of the time needed for a single chess game range from ten minutes to over an hour, depending on the parameters of the game.

The second spiral of our development will be to implement an alpha-beta searching chess-playing algorithm sufficiently similar to Kendall and Whitwell's to make our comparisons to their results valid. Since the authors did not provide exact details of their algorithm, we will use the information they did provide and our other background references to hopefully create a comparable basic individual. In this phase we will leverage our checkers experience from spiral one as well as the basic code architecture that was developed therein.

Once we have completed the re-implementation, we will begin to investigate how the mutation parameters affect the convergence of the evolutionary algorithm. Our basic approach to this will be first to test the basic empirical effect of changing the R values in the mutation equation. If we vary only one parameter at a time and select five different values for each of the four different R weights, this would involve 20 different evolution trials. Given that a single trial may easily take as much as two days or more to complete and that we currently have at most one computer available for running trials, even this seemingly small set of data may be unattainable within our time frame. In such an event, we will attempt to scale our test accordingly, most likely by focusing only on one or two of the R values while leaving the rest constant. This should still provide us with similar data on the relative importance of this smaller set of parameters while demonstrating our competence in implementation and illustrating our powers of analysis given sufficient computational and temporal resources.

Approximate Work Schedule



February 7~23: Complete problem set 1. This problem set became the basis of the remainder of our project. Actually, both Jeremie and Justin reviewed the paper by Kendall and Whitwell which contains the algorithm we plan to re-implement. These weeks were used to gain a background of information on alpha-beta searching and on evolutionary algorithms as they may be applied to chess.

February 23~March 18: Implementation of spiral 1, the original checkers algorithm. This algorithm was a basic alpha-beta coupled with improvements such as heuristic move-ordering, transposition tables, and quiescent search techniques. The basic Kendall and Whitwell algorithm was also implemented for checkers during this spiral. We are very very glad we started the project long before it was actually assigned. Even having spent almost two months on it already, we are feeling a great deal of time pressure to finish in the remaining month.

March 14~April 8: As we were attempting to complete spiral one, we were also required to complete the advanced lecture assignment. This gave us an opportunity to really delve deeply into the Kendall and Whitwell algorithm (even though we had limited time to actually present all of our research.) and also gave us the final impetus to finish our checkers program since it could be double as a very nice lecture demo. The write up for that project is still on-going and will not be completed until April 8.

April 4~April 11: At the current time we are devoting our time not only to the completion of the advanced lecture, but also to the creation of this proposal. The proposal has allowed us the opportunity to examine just how far we have come since February and analyze approximately how much farther we think we will be able to go. We are being very optimistic in our estimates of how much we can actually finish in the next month before the project is due. However, we are both hard workers! ☺ We also have developed the descope plan suggested in the assignment for this proposal, realizing that there is a high probability we may only be able to complete the minimal plan.

April 10~April 23: A very optimistic estimate of the time required to implement the Kendall and Whitwell algorithm starting from the checkers skeleton we have already completed in spiral one. We currently believe this will be a relatively smooth and simple process as we have labored very hard to lay the difficult foundations during spiral one, but as always with implementation, there may be unanticipated difficulties. If we see this task begins to extend too far past April 23 and into May, we will have to adjust the scope of our project accordingly. See the minimal plan below.

April 23~May 9: These approximately two weeks will be used to run our evolutionary trials and collect our data. We are hoping to find more computers on which to run our trials. (If you can help with this, it would be greatly appreciated. Unfortunately we require Windows machines that can be tied up for as much as two days continuously, a rare commodity to come by on campus.) Depending on our computational resources and the amount of overlap we will have with the task before this one, we may be forced to scale our project appropriately. Realistically, the most we can hope to accomplish is that in our baseline plan.

May 8~May 11: We must reserve these last few days for writing up the final report. This will be the period during which we will finalize our data and complete the analysis of our results.

May 12~May 14: Become very drunk/sleep/play lots of video games in a futile attempt to recuperate before finals week! ☺

Descope Plan

In writing this proposal we have realized that our goals may be even more ambitious than we had at first realized, and while we are still hoping for success and willing to do everything in our power to make that outcome as likely as possible, we must accept that some circumstances may not be within our control. Therefore, we are defining this four-level descope plan in order to ensure that we at least are able to reach all of our learning goals and so that our ambition and curiosity does not cause our academic standing to suffer.

Minimal Plan:

- Re-implement the Kendall and Whitwell evolution algorithm for chess. Even if we have time only to run this algorithm once, we would sincerely like to completely re-implement their algorithm in the same context in which it was written.

- Also implement this algorithm in the context of checkers. This should satisfy the project requirements for an extension, namely "create and report upon a novel extension to that approach, reporting on the improvements you make." We would be extending the Kendall and Whitwell algorithm to a previously uncharted domain. This would be our fall-back option if everything went wrong and our projects for our other classes also started going haywire. Hopefully, we can meet these objectives at minimum.

Baseline Plan:

- Satisfy Minimal Plan objectives

- Perform our study of the effects of mutation parameters on evolution convergence in the context of checkers instead of chess. We would still attain the same understanding of all the relevant algorithms and still be able to analyze the effect of mutation parameters on an evolutionary algorithm. This option is very attractive to us because of its reduced amount of computational time. It is not that we are afraid we won't be able to complete the implementation for chess, it is only that the running time of trials may become too cumbersome. We could perform practically the same study we envision for the chess algorithm in a small fraction of the time using checkers. On or

around May 1, we will have to make the decision whether to continue applying our data-taking efforts to chess or to switch to checkers and complete the project in this way. We would be satisfied with this approach and feel it definitely demonstrates a high level of implementation competence as well as a high level of intuitive and novel thinking. The knowledge gained by the technical community would be comparable and the only real difference between this plan and our enhanced plan would be the "flashiness" derived from conducting the study using chess instead of checkers.

Enhanced Plan:

- Satisfy Minimal Plan objectives
- Implement the baseline plan using chess instead of checkers. Again, this will be very difficult to complete considering the amount of computational power at our disposal and the amount of time each evolution trial may require.

Super-Enhanced Plan (i.e. summer project!):

- Perform a meta-evolution. Evolve the mutation parameters themselves. This would involve competing the various evolutions against one another. We would need to develop a method by which to do this. In the simplest form, we could have the product evaluation functions of each evolution compete against the product of another and retain the mutation parameters of only the winning evolution. We think this would be REALLY cool….but sadly there is practically no way we will have the computational resources to complete this. We include the idea just in case one of your Master's students might want to steal it for a thesis or journal article! ☺

References

[1] Graham Kendall and Glenn Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function Using Population Dynamics*. Proceedings of the 2001 IEEE Congress on Evolutionary Compuation, 2001.

[2] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 2003.

[3] Jonathan Schaeffer. The Games Computers (and People) Play, 2000.

[4] T.A. Marsland. *Computer Chess and Search*. Encyclopedia of Artificial Intelligence, 1992.

[5] R. A. E. Makinen, P. Neittaanmaki, J. Periaux, and J. Toivanen. *A Genetic Algorithm for Multiobjective Design and Optimization in Aerodynamics and Electromagnetics*, ECCOMAS, 1998.

[6] William E. Hart, Thomas E. Kammeyer, and Richard K. Belew. *The Role of Development in Genetic Algorithms*, U.C.S.D. Technical Report Number CS94-394, 1994.

[7] Nobuo Sannomiya and Hitoshi Iima. *Genetic Algorithm Approach to a Production Ordering Problem in an Assembly Process with Buffers*. Selected Papers from the 7th IEAC/IFIP/IFORS/IMACS/ISPE Symposum, pages 403-408, Toronto, 1993.

[8] P. Kini, Charles C. Peck, and Atam P. Dhawan. *Genetic Algorithm-based Reconstruction In Diffusion Tomography*. Proceedings of Photon Migration and Imaging in Random Media and Tissue, Los Angeles, 1993.

[9] Sam Sandqvist. *Aspects of Modelling and Simulation of Genetic Algorithms: A Formal Approach.* Helsinki University of Technology Thesis. 1998.

[10] Sebastian Thrun. *Learning to Play the Game of Chess*. Advances in Neural Information Processing Systems 7, 1995.

[11] Johannes Furnkranz. *Machine Learning in Games: A Survey*. Machines that Learn to Play Games, 2000.

[12] Dennis DeCoste. *The Future of Chess-playing Technologies and the Significance of Kasparov Versus Deep Blue*. Papers from the 1997 AAAI Workshop, 1997.