Justin Fox 16.412
2/5/05 PS 1

Part A: Topics of Fascination

The topic I am primarily interested in learning more about is computational cognitive reasoning as it may be applied to game theory, specifically chess. I will never forget the day back in 1997 when Kasparov was actually defeated by Deep Blue. (Although I still contend that Kasparov was paid to throw the match. The evidence for this is that in game 2, he played the computer to a draw, but "didn't realize it" and decided to concede rather than finish the game. IBM NEEDED Deep Blue to win! ☺) From the brief literature survey I conducted on the subject, it appears that most of the mainstream literature has actually tended away from chess-playing in particular over the last few years, though there are a large number of papers from the 80's and early 90's describing the evolution of the minimax algorithm and the many adaptations people have incorporated into it. Aside from the basic search algorithms, more recently neural networks, genetic algorithms, and numerous other machine learning techniques have seen their application to this problem as well. It seems to me that a truly generic, efficient, and novel algorithm for "combinatorial games," as the literature tends to call chess, checkers, and other games of this type, has not really been attempted in many years, with many authors simply biding their time until computers become powerful enough to minimax search the entire game tree. Hopefully something I learn in this class will prove that statement wrong!

A second topic I am interested in is computer vision. Most specifically, I would like to learn the state of the art in visual object detection and recognition techniques. About four years ago as an undergraduate, I worked for two summers at JPL on a project designed to make autonomous Martian rovers that would be as skilled at classifying rocks and geological features as our staff Martian geologist. Our techniques were pretty much state of the art at the time (Gabor filtering for texture analysis, Bayesian classifiers, feature analysis, 3-D stereo data for range and object segmentation), but I was really disheartened by how poorly our methods actually performed in all but certain doctored situations. In a machine learning class at Caltech, we learned about Lowe's object detection algorithm, and I was extremely excited. Then I learned that it too had many of the same major limitations (for instance, changes in illumination direction), and in general did not always work as well as its creators seemed to suggest from their particular demonstrations. I would like to learn more about object classification algorithms and hear if there have been any improvements in the field.

Finally, I would be interested in learning about the use of reasoning techniques as they are applied to the diagnosis of engineering systems. This is probably one of the most useful and practical applications of autonomy at the present time and its improvement and extension will be necessary and profitable in the future. Deep Space 1 demonstrated that a spacecraft could autonomously repair itself in flight, and such algorithms will be necessary on future deeper space missions when communication lags would otherwise severely reduce the efficiency and sustainability of the mission. We learned an algorithm for single-fault detection in 16.413, but I would be interested now in learning the more complex algorithms for multiple faults.

Part D: Researching a Critical Reasoning Method

The first paper I chose to read was *Playing Games with Algorithms: Algorithmic Combinatorial Game Theory* by Erik D. Demaine. In reading this paper, I hoped to attain a general overview of the theory behind combinatorial games like chess, and that is exactly what I was given.

Demaine began by first giving the ubiquitous, but, at least in my practically ignorant case, extremely helpful introduction to and definition of combinatorial games. These games involve two players, named Left and Right, who alternate turns using deterministic moves. One of the most important aspects of this class of games is that they are by definition *perfect information* games, that is both players know all the states of the game at every instant, both players know what moves the other player has made, and all actions and consequences are completely deterministic and observable. While these restrictions are sufficiently loose to incorporate innumerable common games such as chess, checkers, and Go, they are restrictive enough to allow game theoreticians a handle by which they can manipulate this subclass of problems.

With the problem well-stated, the author then continued by introducing the most important game theoretical technique used in analyzing the computational complexity of problems associated with combinatorial games, Conway's Surreal Numbers. A surreal number is actually defined by a pair of sets {L|R} in which all numbers in L are less than all numbers in R. The value of this surreal number is then "the 'simplest' number larger than all Left options and smaller than all Right options." While these numbers may seem to be purely a demented mathematician's idea of a strange joke, it turns out that they can be very useful in making analogies to combinatorial games. Demaine explained that games with a surreal value of 0, a positive number, or a negative number have the predictable outcomes that the second player to move wins, the player named Left wins, or the player named Right wins, respectively. Those games, such as {1|0} outside the set of surreal numbers have the predictable outcome that the first player to move wins. The paper goes on to explain a number of other ways in which surreal numbers have been used to prove numerous results regarding the predictability and complexity of combinatorial games.

Finally the paper analyzed a few well-known combinatorial games, including chess, with respect to their expected computational complexity. It was shown that given an arbitrary chess board configuration, it is EXP-TIME complete to determine the winner. This implies that there can be no polynomial time algorithm for computing the winner of a general chess game no matter how hard we try to think of one, a fact which is important to know so that we do not waste too much time on our project!

Overall, this first paper was meant to be a general introduction to the basics of game theory and some of its applications to chess and other games like it. In this respect, it served its purpose well. However, with respect to the other papers, I felt that for my purposes this one might have been slightly too theoretical, with not enough practical applications to truly be of use to my present project. The tidbit regarding the exponential

complexity of determining a winner in chess was helpful however, and I will remember that trying to look for a checkmate even a few moves ahead is inherently an inefficient task.

The second paper I read was *The Games Computers (and People) Play* by Jonathan Schaeffer. This was actually a 70 page "chapter" written as an update to Arthur Samuel's well-known 1960 chapter in the textbook *Advances in Computers*. I chose to read it because in browsing it, I noticed that not only did it explore in-depth various search methods and their extensions, but it also exhibited a fairly detailed analysis of the Deep Blue chess program and its match against Kasparov, making it a great basis for the present project.

The main focus of the first section of the paper was in describing the mini-max and alpha-beta game tree search algorithms. We briefly covered these algorithms in 16.413 and I have already used alpha-beta searching to create my own checkers program, so much of this section was review. What I found extremely interesting and useful, however, was that the author not only described the basic algorithm as Russel and Norvig did, but also detailed many of the improvements that had been made in recent years to the simple application. In particular, Schaeffer discussed four techniques that have been used to speed up or otherwise improve alpha-beta search, describing these techniques as "the recipes for building high-performance games." The first of these techniques was caching. In games where multiple sequences of moves may evolve the board into the exact same state, storing the results of previous searches in a hash table can have dramatic effects. Schaeffer quotes as much as a 75% reduction for chess searches and an 89% reduction for checkers. A second method for reducing the cost of an alpha-beta search is move ordering. The benefits of pruning are realized most dramatically when the search is able to cut off a branch after examining only a very small number of its leaves. Schaeffer describes a number of ways in which this ordering can be performed, including the use of the cached transposition tale and application dependent heuristics. The third method for improving alpha-beta search was guessing a smaller search window. That is, at each iteration, alpha and beta are normally initialized to be [-inf,inf]. However, these extreme values are rarely realized and reducing the range of the window can greatly enhance the search speed. Schaeffer described this as a "gamble" because reducing the window too much may require another search to be performed, but he also described a number of heuristics for intelligently choosing the search window which would minimize the risk of this gamble. The fourth and final method described by Schaeffer he described in financial terms. If you are investing in stocks, you put more money in the stocks you think will perform well and less in the stocks you believe will perform badly. Similarly, when you search, you search deeper in branches that seem to be getting somewhere and cut off early branches that seem to be going nowhere. He mentioned a number of heuristics for doing this in chess in particular, many of which may be directly applicable to our final project.

The other section of the chapter that I found very interesting was Schaeffer's description of Deep Blue's algorithm and its performance against Kasparov. Deep Blue was the result of a decade of IBM funded research and was made up of parallel processors, dedicated chess chips, and the consultation of a chess grandmaster, Joel Benjamin. Schaeffer, I feel, accurately appraised the significance of the match, saying

that it is only one intriguing data point with no statistical significance and that Deep Blue's construction was, while definitely an improvement in computer chess-playing, largely unproductive since it was immediately dismantled and its funding completely cut. The analyses contained in this section highlight some of the important pitfalls typical computer programs fall into that humans are able to avoid and will be priceless when analyzing our own algorithm.

I am really glad I found this paper. It will be invaluable for our project. We could even just compare the efficiencies and performances of the four types of methods described in the paper just to give a quantitative understanding of how each one affects the search. Compared to the other papers, it was broad and basic enough to be useful and understandable, and practical enough to be useful in an applied sense.

The third and final paper I chose to read was entitled *An Evolutionary Approach for the Tuning of a Chess Evaluation Function Using Population Dynamics* and was written by Kendall and Whitwell. After reading Schaeffer's review of the Deep Blue algorithm, it became apparent that the most important part of a computer chess player's performance is often its evaluation function, and these authors presented an interesting approach to developing such a function.

Computer chess programs have historically used a linear combination of weighted parameters to heuristically evaluate the "goodness" of a particular board variant. The general purpose of this paper was to create a method whereby a computer chess evaluation function could be created without a human having to hand-tune each of the individual weighting parameters in the function. This was done through the use of an evolutionary algorithm. Basically, a particular evaluation function was considered, namely a weighted sum of the player's number of kings, number of queens, number of rooks, number of bishops, number of knights, number of pawns, and the number of legal moves available to the player. Fifty such functions were initially randomly generated with randomized weight values. These functions were then inserted into a chess program that played the two candidates as opponents. The losing function was removed from the population. Whichever function won the match was then slightly mutated and replaced in the population. It was also allowed to "breed," and another slightly mutated copy of it replaced the losing function. In this way, new functions were evolved which tended to play better and better chess. Of course, the paper described a number of details involving the need to account for local minima and the algorithms used to mutate and compete the various functions. Once the function population had reached quiescence, their final values were averaged and taken to be the evolved evaluation function.

The authors tested their initial evaluation function against the final evolved one using a commercial computer game that gave ranking information regarding the player's skills. The undeveloped function lost within 60 moves. The developed function was actually able to win one game and lost the other game in almost 100 moves, showing quite vividly that the evolution process had created an improvement. In addition, the developed function was rated a 1750 on a scale that Garry Kasparov was rated a 2805. The undeveloped function, meanwhile rated a meager 650. Clearly the evolutionary approach provides a fast, simple method for creating the weights of a computer chess evaluation function which does not need human intervention to optimize.

I had always heard about evolutionary methods and understood more or less how they worked. This is, however, the first paper I have actually read that detailed the equations, the mutation probabilities, and such. I found it to be extremely interesting, clearly written, and hopefully useful. I am even considering "evolving" a checkers evaluation function in my spare time to beef up my own autonomous player. I think this would definitely be an option in selecting an evaluation function for our project though I'm afraid five weeks may fly by faster than I would like! ☺

Part E: A Simple Project For Your Cognitive Robot

As I mentioned earlier, an autonomous general for the military is a bit out of reach at the moment. Therefore, I would like to implement an autonomous chess player that would demonstrate that computers could at least reason capably in a "simulated war" and that tactics and strategy could be well within the reach of future generations of machines.

We would probably create a simulation in Visual Basic, perhaps researching, understanding, and implementing one or two of the various algorithms currently available and maybe even finding some way to add our own personal tweak to them. We could hopefully then play the algorithms against one another to benchmark their relative performances. If this project is acceptable, I think it would be pretty likely that I would choose to work on it.