



What A Way to Go

By Suzanne Ross
December 13, 2004 12:00 AM PT

Go is more than a game. It spans centuries, it's integrated into religion, politics, and business, it's been embraced by poets and warriors, and it's considered one of the great artificial intelligence challenges today.

[Thore Graepel](#), a scientist in the [Machine Learning and Perception](#) group at Microsoft Research Cambridge, and a Go player with the status of first Dan, has spent many years contemplating a computer program that could beat even an average Go player. To date, this has never been done.

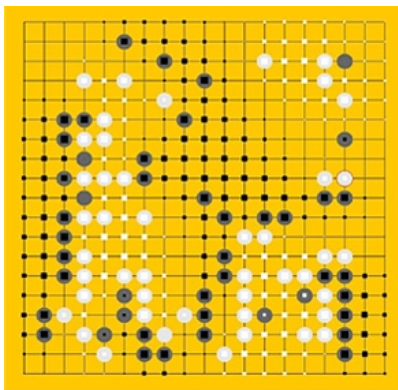
Graepel, along with David MacKay, a professor, and David Stern, a PhD candidate, both from the University of Cambridge, will present their work at the December 2004 conference on [Neural Information in Processing Systems conference](#) in Vancouver, Canada.

"Computer Go is one of the great AI challenges today. Starting from the 1950s it was chess, but after Kasparov lost against IBM's Deep Blue computer, a lot of people lost interest in it," said Graepel.

"However, current computer Go programs can hardly beat even mediocre amateur players," explained Graepel.

Go has several levels of skill. The amateur rankings range from 30 kyu to 1 kyu (student ranks) followed by 1 dan to 7 or 8 dan (master ranks). Professional levels go from 1 to 9 dan.

Go looks like a simple game on the surface. It's played on a rectangular board with 19x19 intersections, creating a total of 361 intersections. There are two players - black and white. At each move, a player places a single stone on one of the intersections. The object of the game is to gain as much territory as possible, while preventing your opponent from gaining territory or capturing you. The art and complexity is in how you accomplish this.



This is the middle position of a Go game. Overlaid is the estimate for the probability of becoming black or white for every intersection. Large squares mean the probability is higher.

Though there's controversy about the origins of the game, most people agree that it started in China sometime between 2,500 and 4,000 years ago. It has sparked interest in everyone from mathematicians to psychologists to business leaders, and been credited with increasing our knowledge of combinatorial game theory.

To devise a computer program that will play chess against a human, you define a function that tells you how good a given position is for one player. Then you specify all the positions that can be reached with one move and all the positions that can be reached with a given response of the opponent. The computer crunches through millions of positions and

evaluates all of them to pick the best move.

That brute force method doesn't work in Go, because there are about 200-300

possible moves you can make when it's your turn. In the first fourteen moves in a Go game, the game tree has 10 thousand million leaves. The program that beat Kasparov could take over one and a half years to play one move in Go.

"So it's infeasible to enumerate all my possible moves and for every one of them enumerate all the possible moves of my opponent and so forth, because there's just too much complexity," explains Graepel.

"Our argument is that the big game tree that results from enumerating all the moves the two players could make induces uncertainty in the mind of the player, because of its complexity. The player doesn't know what his opponent might do, and he doesn't know how his own moves will affect the future developments of the game. His computational resources aren't large enough to analyze the game tree. So we believe that one has to explicitly model this uncertainty and that an excellent way of modeling uncertainty is using probability theory."

Usually, probability theory is used in non-deterministic settings, such as weather forecasting or games of chance such as black jack. But here the researchers are using it in a fully deterministic setting to model the uncertainty involved. Because a player can see the full extent of the board position, and there is no roll of the dice, it is considered a deterministic and fully observed setting.

"The difference between the size of black and white territories is just the outcome of the game - who wins. If we could estimate that at a given point in the game, we would have solved the problem of evaluating the position, and we could then use techniques similar to chess to solve the game," said Graepel.

"Since we can't determine exactly which points will be black or white we aim at estimating the probability of them being black or white.

"We make a model for the probability of a given configuration of white and black territory - such as one half of the board being black and the other half being white. Another, less likely, configuration would be a checkerboard pattern of black and white - there are many configurations.

"We have a database of 22,000 recorded Go games played by professional players in China, Japan, Korea and Taiwan. It's a very traditional and honorable profession, particularly in Japan.

"From the perspective of computer Go, these games can be considered perfect. The nice thing is that for each of these games, if you jumped into the middle of the game at a given position, you can pair this observation with the known ending. So for every point on the board we know if it turned out to be white or black territory and that's the training data.

"We pair the opening position with the known ending and we learn the parameters of our model. We feed the training data into our model and see which predictions it makes for each point.

"We compare that to the outcome that was achieved and adjust the parameters so that the prediction of the model comes closer to the actual outcome of the game.

"And in that way we capture the essence of the 22,000 games in our model - within the constraints of the model.

"It gives us a model that we can apply to new situations that haven't been encountered. Then the model can determine the probability for every point on the board of becoming white or black territory at the end of the game. And if I sum over the probabilities for every point on the board it gives me an estimated score for the game to see who would win - black or white. Then you can use the expected outcome as a building block for an evaluation function."




Though Graepel and his colleagues haven't written a fully functional program that can hold its own against a human, they believe that their approach could contribute to such an effort. They need to design several other modules that will work with their current module to cover all aspects of the game. They are pursuing this research and considering other applications for their work.

Other applications for the research into modeling uncertainty in the game of Go are found in the field of computer vision.

"If you take a Go position and you want to segment it into what will become black territory and what will become white territory, it is very akin to certain problems in computer vision. In computer vision, similar models are used to segment images. For example imagine you have an image of a horse in front of a landscape and you would like to determine which pixels belong to the horse and which pixels belong to the landscape, so that you can separate the horse from the landscape. The analogy would be horse corresponds to black territory, the landscape corresponds to white territory, or vice versa. In Go the model takes into account the board position, in the image it is based on the color and texture information.

"The Japanese have a word, *aji*. Literally translated, it means "taste." Taste lingers, and so does the influence of a Go stone, even if it appears weak or dead, because of the influence it can have in the future," said Graepel. "It is this lingering and subtle effect of stones that creates uncertainty and makes probability theory an appropriate language for talking about Go."

Modeling the game of Go is akin to modeling one of humanity's most puzzling mysteries - what does the future hold?

 Share  E-mail this  Print

[Back to top](#) 