Evolution of a Digital Organism Playing Go

Christian Alt and Helmut A. Mayer
Department of Computer Sciences
University of Salzburg
Salzburg, Austria
christian.alt@stud.sbg.ac.at, helmut@cosy.sbg.ac.at

***Abstract*—Digital organisms (DOs) model the basic structure and development of natural organisms to create robust, scalable, and adaptive solutions to problems from different fields. The applicability of DOs has been investigated mainly on a few synthetic problems like pattern creation, but on a very limited number of real world problems, e.g., the creation of architectural structures. In this paper the potential of DOs for learning to play the game of Go is demonstrated. Go has been chosen for its high complexity, its simple set of rules, and its pattern–oriented structure. A DO is designed, which is able to learn to play the game of Go by means of artificial evolution. The DO is evolved against three computer opponents of different strength on a 5×5 board. Specifically, we are interested in the DO's scalability, when evolved to play on the small board and transferred to a larger board without any external adaptations.**

I. Introduction

The game of Go has gained increasing attention by computational intelligence research since *Deep Blue* has beaten the reigning world champion in chess, *Garry Kasparov*, in 1997, which marked the beginning of the dominance of chess computers over human players. Though, the rule set of Go is very simple, the game's complexity is far beyond that of chess. It, therefore, offers a (new) challenge for various computational intelligence approaches.

Because of the pattern based nature of Go, its inherent symmetries, and its scalability (Go is played on boards of various sizes), we considered DOs to be a good choice, as they might be able to exploit the symmetric and scalable structure of the game. Of course, details of the DO have to be designed such that it captures essential properties of the game. The basic ideas can be summarized in a few sentences.

The DO's environment is the Go board and its intersections. An intersection may be "inhabited" by a single cell indicating the state of the intersection (empty or black/white stone). All the cells on the board communicate by diffusing and absorbing *mediators*, which in turn may change the cell state. The activities of a cell are governed by the *cell program*, which is evolved and controls the cell state mainly based on mediator concentration. E.g., a cell may replicate and/or change its state. The latter may lead to a stone placed on that intersection (cell). It is important to note that in this work we are only concerned with DOs having identical cell programs, i.e., each single cell is controlled by the exact same set of instructions.

When a DO is employed to play the game of Go its intelligence is distributed across the board in the cells of the artificial organism. There is no central entity coordinating the cells. This decentralized organization leads to a decoupling of search space and solution space, which might enhance the emergence of more general playing abilities, e.g., the symmetric transformations of sequences played in different corners of the board. The latter is quite simple for humans, but poses great difficulties for a number of computational approaches.

A strong motivation for the application of DOs in this context was the assumption that a DO may scale well, when transferred to a board of different size, which, again, is a substantial problem for other approaches (e.g., artificial neural networks). As the computational cost for evolving a DO decreases with board size, the projected scalability could result in a decent player on larger boards, which has been evolved on smaller boards in a moderate time span.

II. The Game of Go

Go is a two–player board game that is played on a board with a grid of horizontal and vertical lines. Usually, the board size is 19×19, but there are smaller sizes often used for educational purposes ($9 \times 9, 5 \times 5$). The two players take turns in placing black or white stones on the intersections (of the lines) on the board. The black player (usually) starts the game with the first stone put on the board. Once a stone is placed it stays at its position for the whole game, unless it is captured by the opponent.

Two or more stones of the same color that are placed on intersections adjacent to each other form a group. Empty intersections next to a group are called liberties. If a group's number of liberties is reduced to zero it is captured and removed from the board. Players are allowed to pass, if they do not deem a next move to be beneficial. The game ends when both players pass in consecutive moves.

The goal of the game is to secure territory (empty space surrounded by stones of the same color). Captured stones also improve the final score of a player, which basically is the sum of territory points and the number of stones captured by a player. In order to compensate the advantage of the black player making the first move, the white player receives bonus points called *Komi* (usually 4.5–6.5) added to his score.

For calculating the players' final scores two widely used methods, namely, *Chinese* and *Japanese scoring*, are used. With Japanese scoring the number of empty intersections in a player's territory are added to the number of stones captured by the player. When employing Chinese scoring a player's territory points and the number of stones on the board are added. The prisoners are not taken into account because they are implicitly given by the number of (live) stones on the board. With a few exceptions the outcome of a game is not dependent on the scoring method.

Go has a sophisticated ranking system rating the players' strength. The amateur ranks are called *kyu* starting from about 35 kyu going up to 1 kyu. The higher ranks for expert (amateur) players are called *dan* ranging from 1 dan to 7 dan.

Even above are the professional ranks from 1 dan to 9 dan. A detailed introduction to Go can be found in [1].

III. Related Work

A number of different approaches to design computer programs that are able to play the game of Go at a reasonable level have been investigated. Todays, most competitive Go programs are modeling human expert knowledge or use Monte Carlo methods, which are simple, but require ample computational power.

GNU Go [1] and *The Many Faces of Go* [2] are two of the best known handcrafted computer Go programs. Because of the huge success of Monte Carlo methods, both programs also incorporate these techniques. Lee et al. (2009) developed a very successful Go program called *MoGo* combining Monte Carlo tree search evaluation, moves extracted from databases, and an expert rule system [2].

There have also been several approaches using *Artificial Neural Networks* (ANNs) to either assess the current board situation or to choose the next move directly. Schraudolph et al. (2000) presented an approach using *Temporal Difference Learning* to evaluate board situations [3]. Dahl (1999) presented an approach using artificial neural networks combined with *Alpha–Beta Search* (for connectivity and life–and–death problems) and a *Joseki* (corner openings) database [4]. A similar approach integrating a priori expert knowledge into an ANN playing Go was published in [5].

Mayer (2007) presented a comparison of different board representations in the input layer of ANNs learning by temporal difference [6]. Because neighborhood is a very important concept in Go, the author investigated three different board representations, each with a different level of neighborhood information for the neural network.

Richards et al. (1997) presented an approach to evolve Go–playing neural networks using *Symbiotic Adaptive Neuro Evolution* (SANE) [7]. A coevolutionary approach to generate neural Go players was published in [8]. The authors pointed out potential advantages of coevolution, as there is no need for a pre–defined opponent, and the level of play reached may not be limited by the quality of a specific opponent. Mayer and Maier (2005) presented a cultural approach of coevolution of ANNs playing Go [9], where the "knowledge" of good players of previous generations is (implicitly) stored.

IV. Digital Organisms

In nature very complex organisms consisting of millions of cells are grown from a considerably smaller, less complex genotype. These organisms exhibit certain desirable characteristics like scalability, adaptability, robustness, and the ability of self-repair and recovery. Digital organisms (DOs) model the basic concepts of natural organisms with the goal to generate problem solutions exhibiting above properties, as these are also desirable in a non–biological context.

A DO is composed of *cells* in an *environment*, which communicate by the emission and absorption of *mediators* controlled by the *cell program*. The *cell state* represents the "decision" of a cell, which has to be defined in the context of a specific problem. The cell program also controls various actions of the cell, e.g., replication, growth, differentiation, movement, or performing programmed cell death (more profanely, dying).

[1]http://www.gnu.org/software/gnugo/

[2]http://www.smart-games.com/manyfaces.html

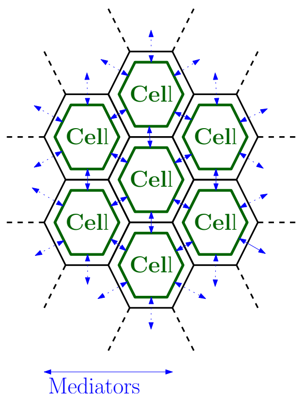


Figure 1. An exemplary digital organism with hexagonal cell structure.

In contrast to *Cellular Automata* (CA), the number of cells in DOs is dynamic, whereas it is static in CAs. They also differ in the cells' connectivity patterns. In CAs cells usually have fixed connections to other cells, while DOs have no explicit connections, but are influenced by mediators, which may be diffused from any other cell. Hence, all cells of a DO are implicitly connected to various degrees.

The cell program is subjected to evolution and its fitness is determined by the problem solving capacity of the DO constructed and controlled by the cell program. This is an indirect form of artificial evolution, as the structure of the DO and its cell states are the result of the (same) cell program acting in each cell.

The environment and the possible shapes and neighborhood structure of the cells have to be designed according to the problem to be solved. The neighborhood structure is important for the diffusion of mediators in the environment. In figure 1 a two–dimensional environment with hexagonally shaped cells is depicted as an example. Each hexagonal cell has six neighbors, which defines the diffusion pattern of mediators released by single cells.

The development of a DO starts with the placement of an initial cell (*zygote*) into the environment. Then, the control program of the cell is executed, which usually leads to cell replication. During evolution a control program, which does not enhance replication, will most likely be deselected (depending on the problem to be solved). A *cycle* is defined by the complete execution of the cell program of each cell of the DO. In one cell the cell program may cause the movement of a cell, while in another cell the identical cell program may lead to a release of mediators, and in yet another cell the cell program may change the cell state. It is this complex interaction of cells, cell positions, mediators, cell states, and the cell program, which is responsible for the potentially "intelligent" behavior of a DO.

Again, it should be stressed that the final DO behavior is the result of an evolutionary process, in which the control program is constantly reshaped and evaluated by observing the behavior of the DO it is controlling. For the evolution of the control program we employ *Genetic Programming* (GP), which will be briefly reviewed in the following section.

A. Genetic Programming

GP is an evolutionary technique adapted to generate computer programs (or algorithms), which loosely spoken leads to a computer programming itself. John Koza (1994), the founder of GP, stated that "in genetic programming, populations of computer programs are genetically bred using the Darwinian principle of survival of the fittest and using a genetic crossover (sexual recombination) operator appropriate for genetically mating computer programs" [10].
GP starts with an initial population of randomly generated computer programs, over many generations selects the better programs based on their performance on a set of training examples, and applies genetic operators in order to modify the programs. Various genetic operators categorized into primary and secondary operators have been proposed [11]. Primary operators are reproduction and crossover. Secondary operators are mutation, permutation, editing, encapsulation, and decimation. The computer programs evolved using GP can be simple mathematical functions, rule bases, fuzzy systems, different assembly language programs, or even high level language programs (e.g., LISP programs). Programs are encoded as a syntax tree similar to a parse tree generated by most compilers. The syntax tree is a rooted, node–labeled tree. The root node and the internal nodes are populated by components from a function set, while the leafs of the tree contain components from a terminal set. This representation of a computer program is of variable size and shape.
A specific GP variant is *strongly typed GP* [12]. It employs a type system, where each terminal has a given type (e.g., an integer number) and each function has a return type (whose value is propagated to the parent node) and parameter types (whose values are received from child nodes). The types of the different nodes have to be taken into account during the initial generation of syntax trees as well as during crossover and mutation. Hence, it can be guaranteed that every resulting tree represents a syntactically correct program.
In our work strongly typed GP is used to evolve a rule base representing the cell program. Figure 2 shows the syntax tree of a single rule.

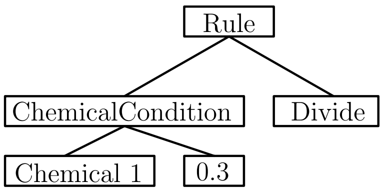


Figure 2. An exemplary cell program rule in syntax tree representation.

With this particular rule a cell is replicated, hereby creating a new cell, if the concentration of `Chemical 1` is larger than 0.3. A variable number of rules of this structure makes up the rule base, i.e., the complete cell program.

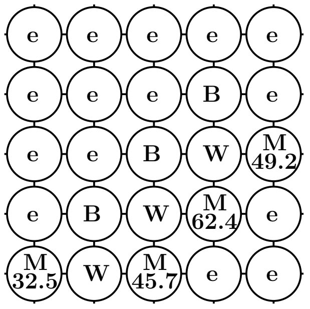
B. Applications of Digital Organisms

A few researchers have employed DOs for mostly synthetic problems. Miller [13] presented an approach using square cells in a two–dimensional grid environment and feed–forward boolean switch circuits that were evolved using cartesian genetic programming. In his work the evolution of a self-repairing organism representing form and colors of the French flag was investigated.
Roggen and Federici (2004) evolved patterns of different size ranging from 8×8 to 128×128 cells [14]. A variety of similar experiments was presented in [15] and [16].
A real–world application of DOs is the creation of stable architectural structures. Kowaliw et al. (2007) used an approach similar to cellular automata to evolve a DO solving that task [17]. The DOs were grown in two-dimensional environments with different sizes and shapes.

C. A Digital Organism playing Go

Until now, DOs have not been used in the domain of board games. However, they might be very proficient in this domain, specifically, with pattern–based games like Go. Certain specifications and restrictions have to be made to adapt the DO mechanics to the game of Go, which are described in the following.
When using a DO for playing Go, naturally, the Go board and its intersections are the DO environment, where at each intersection a cell may live. This basic setup introduces a property not found in most other computational intelligence approaches, as game playing is not controlled by a central entity, but intelligence is distributed all over the board in cells with identical cell programs.
Each Go cell is defined to have one of four cell states. The state `Empty` indicates that this intersection is not considered for a move, while state `Move` makes the intersection a candidate for a move. The latter two states can be changed by the cell program, but the other two states `OwnStone` and `OpponentStone` are forced by the rules of the game, i.e., stones having been placed already cannot be moved (they are removed, if captured). Hence, when the DO is playing a game against an opponent, a move from either player results in the change of the corresponding cell state. E.g., a cell may have had the state `Empty`, but when a stone is placed on its intersection, the state is forced to represent this stone. If there is no cell on the intersection, a cell representing the stone is created.
Note that with this scheme there is no need to convey the current board situation to the system, as the DO **is** essentially the current board situation. When the DO player is asked to generate a move, it considers all cells with state `Move`, determines the cell with largest size, i.e., move quality, and places a stone at the corresponding intersection, hereby, changing the cell state to `OwnStone`. The cell size is regulated by the cell program. Again, there may be some benefit with this move generation scheme, as there might be only a few `Move` cells representing a small set of potentially good moves. If there are no `Move` cells on the board, the DO issues a pass move.
An example of the states of Go cells during a game is depicted in figure 3, where each intersection is covered by a cell.

Three own stones marked with B and three opponent stones marked with W are on the board. Four cells are in state `Move` indicating suggestions for the next move. The cell size given with the move cells correspond to the (assumed) move quality. Here, the intersection of the move cell with largest size (62.4) would be selected for the next stone to be played (a nice double–atari move in this case).
The definition of neighborhood in the environment naturally



e	Empty
B	OwnStone (Black)
W	OpponentStone (White)
M	Move

Figure 3. Cell states of Go cells in an exemplary board situation.

resembles the Go neighborhood concept. Only cells that are on horizontally or vertically adjacent intersections are considered to be neighbors, as stones build a *group* in Go, if they are neighbors in the above sense. Hence, any cell has a maximum of four neighbors, which absorb the mediators potentially released by the cell.
The various steps of a complete game are depicted in figure 4 and described in the following.

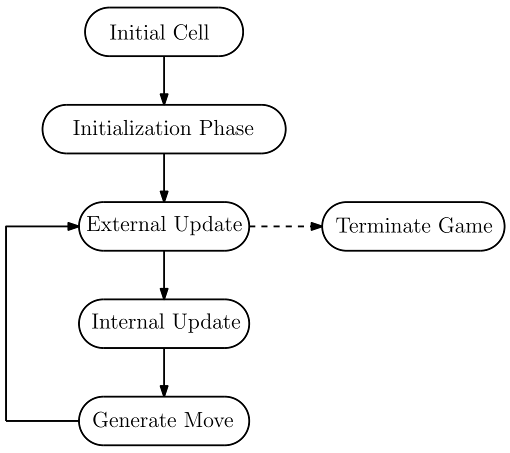


Figure 4. Flow of digital organism procedures for a complete game.

- **Initial Cell** – The initial cell (zygote) is placed in the environment, either at random, or at a specific position.
- **Initialization Phase** – The initialization phase is carried out before the game is actually started. A number of (initial) cycles is executed so as to grow the DO, which could be utilized to explore the board size and/or to locate the center of the board.
- **External Update** – The current move is "injected" to the DO. The corresponding cell (state) is set and (potentially) captured stones (cells) are set to `Empty`. In this step the cell states are externally updated, but not by the cell program, in order to reflect the move made by the DO or the opponent.
- **Internal Update** – Here a number of (regular) cycles is executed to compute the reactions of the DO to the last move. Now the cell program is controlling the DO in the usual manner.
- **Generate Move** – The DO is selecting the next move out of all cells in state `Move`. These move cells are ranked according to their cell size (move quality). If different move cells share the highest value, one of these moves is selected randomly. This implicit randomness is beneficial during DO evolution, as game variation is increased.
- **Terminate Game** – As given by the rules, the game ends on consecutive pass moves of each player. The DO generates a pass move, if no cells in state `Move` are present.

In order to evolve the rule base for the cell program using strongly typed GP an appropriate function and terminal set have to be defined. Every rule has a condition and a cell function that is executed if the condition is true, i.e., rules are of the form `if` *condition* `then` *cell function*. The function set contains conditions and cell functions, and the terminal set consists of parameters and (some other) cell functions. A variable number of rules is encoded in the program tree. The rules are constructed using only elements of the following function and terminal set.

The function set contains

a `boolean` *CellStateCondition*(`CellState`) returning, if the cell is in the given cell state.
A `boolean` *CellSizeCondition*(`real`) returning, if the cell size is above the given parameter.
A `boolean` *MediatorCondition*(`MediatorType, real`) returning, if the concentration of the given mediator type is greater than the given threshold.
A *ChangeCellState*(`CellState`) changing a cell's state to the given cell state.
A *ChangeCellSize*(`real`) changing the size of a cell to the given size. When a cell is in state `Move`, the cell size is interpreted as move quality.
A *ReleaseMediator*(`MediatorType, real`) generating the given amount of the given mediator in a cell. In each cycle the mediators are diffused to the cell's neighbors according to a pre-defined *diffusion rate*.
A *Move*(`Direction, integer`) moves the cell in the given direction for the given distance. E.g., a distance of 1 moves the cell to the an (empty) neighbor position.
A *ReplicateTo*(`Direction, integer`) is similar to `Move`(), however, the moving cell is a cell clone, i.e., the cell stays at its position and the clone is moved to a different position.

The terminal set contains

a *Replicate*() creating a cell clone and moving it to an empty position closest to the cell.
A *Die*() removing the cell from the DO leaving an empty position.
A `CellState` representing one of the pre–defined cell states.
A `MediatorType` representing one of the pre–defined mediator types. These types are only defined to have different names. The "meaning" of these mediators is defined by evolution, which is free to use only a subset of mediators.
A `Direction` representing one of North, East, South, and

West.
Types `boolean`, `integer`, and `real` are the known primitive types.

V. Experimental Setup

The experimental setup evolving a DO against a number of computer players on a 5×5 board is described in this section. In the following we will call a (fully) evolved DO *diGO player* or shortly *diGO*.
The DO is evolved using a multi–population approach (e.g., [18]) with a total population size of 3,000 individuals, which are split into 30 sub–populations with 100 individuals each. The sub–populations are arranged in a ring topology where the best five percent of each population migrate to the neighboring sub–population in every fifth generation. The migrating individuals replace the worst individuals of the target population. Elite tournament selection is used with a tournament size of two. The number of generations is set to 400, and each evolutionary run is repeated 20 times.
The GP population is initialized with program trees with a maximal depth of four. The crossover probability is set to 0.95 and mutation (random subtree insertion) occurs at a rate of 0.01. The initial `real` type values are drawn randomly out of $[-1.0, 1.0]$, and are mutated with random values from the normal distribution $N(0.0, 0.0075)$. This additional (node) mutation with a rate of 1.0 affects only the `real` terminal.
The DO can use five different mediators for cell communication. The mediator diffusion rate is set to 0.5, i.e., 50% of the mediator amount is diffused to the neighbors. The mediator *reduction rate* is set to 0.3. The reduction rate models the cell's internal consumption of a mediator.

A. Computer Players

Three computer players with different playing abilities, which are described in the following, are used as opponents in evolutionary experiments. Additionally, the well–known and strong GNU Go is used to assess the playing abilities of diGO.

- **Random** – This player chooses a move randomly from the set of possible moves (including the pass move).
- **Naixt** – This (naive) player has very basic playing abilities and employs elementary concepts of Go, i.e., it tries to save own stones, capture opponent stones, and gives up on stones it considers to be dead.
- **GOjen** – GOjen is based on Fuming Wang's *JaGo* program and employs some standard Go playing techniques, e.g., it searches the board for 32 well–known patterns and their symmetrical transformations. If a pattern is detected, it responds with a move stored with this pattern.
- **GNU Go** – This open–source project player mainly employs a variety of strategic and tactical analyses based on human knowledge to improve the quality of play. Its latest official version 3.8 is ranked between 5 kyu and 3 dan on the *KGS Go Server* [3]. Though, it would be a very interesting opponent during evolution, in this work it is only used in test games against diGO, as GNU Go's computational cost is quite large (even on small boards).

[3]http://www.gokgs.com/graphPage.jsp?user=GNU

B. Experiment Configurations

In experiments three different configurations (Table I) are investigated differing in number and type of opponents and number of games played for fitness evaluation.

Table I. Experiment configurations.

Configuration	**Opponents**	**Games**
1N	Naixt	32
2RG	Random, GOjen	16
3RNG	Random, Naixt, GOjen	12

The DO to be evaluated plays half of the games against each opponent with the black (white) stones. The white player always receives a komi of 4.5. For determining the outcome of games Japanese scoring (Section II) is used. The fitness is the win percentage of the DO against the opponent(s).
With configuration *1N* the only opponent is the Naixt player playing 32 games against each DO so as to assess the fitness. With configuration *2RG* both, GOjen and Random, are used as opponents. The Random player enhances a discrimination of the DO's playing abilities in early generations and helps to maintain game diversity. The latter makes evolution less prone to convergence to a diGO player winning many games with identical move sequences, which is more likely to happen against "strategic" computer players like GOjen. For fitness evaluation a DO plays 16 games against each opponent resulting in a total of 32 games.
With configuration *3RNG* all three computer players are challenging the DOs during evolution. This should create a large variety of different games potentially resulting in the most versatile players. Twelve games are played against each opponent to evaluate the fitness.
In order to assess the quality of a diGO player a *strength* (value) is defined by the win percentage against the computer players, when an equal number of games is played against each of the three opponents. The games played to assign strength values are called *Test Games*, as they are not part of the evolutionary process. As each experiment is run 20 times, the minimal, maximal, and average strength of 20 diGO players (the best from each run) is given in the next section. Also, the scalability of the diGO players is assessed by letting the exact same diGO evolved on the 5×5 board play on 7×7 and 9×9 boards.

VI. Experimental Results

In this section the results of the experiments are presented, and a few selected games of diGO players are analyzed in detail. A single evolutionary run took 104 hours on average on an Intel Xeon 2.27 GHz processor.

A. Evolution against Naixt

The results of the experiments evolving diGO players against the Naixt player (Configuration 1N) are shown in table II.

Table II. TEST GAME RESULTS (WIN PERCENTAGES) OF THE BEST DIGO PLAYERS EVOLVED AGAINST NAIXT.

	Random	Naixt			GOJen	Strength
	5×5	5×5	7×7	9×9	5×5	5×5
Min	0.748	0.359	0.03	0.00	0.000	0.410
Avg	0.909	0.471	0.17	0.08	0.130	0.504
Max	0.991	0.691	0.44	0.37	0.271	0.601

The evolved players exhibit a maximum strength of 0.601 and an average strength of 0.504. When increasing the board size, the average win percentage against Naixt drops sharply from about 47 percent to 17 percent. However, specific diGOs do scale better, as can be seen by the maximum win percentage against Naixt. A specific diGO kept its playing level starting at 0.512 on 5×5 and winning 44.1 percent of all test games on 7×7 and 37.2 percent on 9×9. Considering that the Naixt player gets stronger on these medium sized boards, it seems that most playing abilities acquired by diGO in evolution on the 5×5 board are still present on the larger boards.
The win percentages against GOJen in this configuration are reasonable, as the latter computer player is ranked above Naixt. The average win percentages against Random (0.909) and GOJen (0.130) show that the diGO players are not overly specialized to the opponent presented in evolution.

B. Evolution against Random and GOJen

In figure 5 the average and maximum fitness during evolution is depicted over the number of generations.

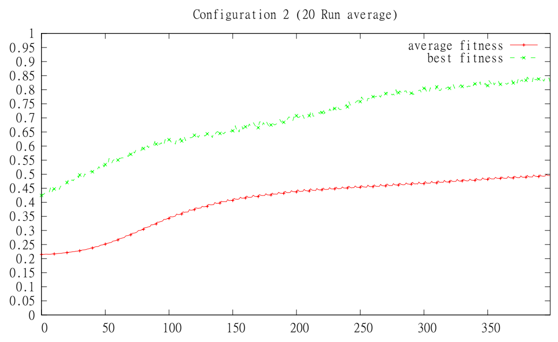


Figure 5. Best and average fitness in evolution against Random and GOJen (averaged on 20 Runs).

Both fitness measures are increasing continuously still having an upward trend in the final generations. The small spikes in the average fitness graph mark the generations in which migration occurred, where a portion of individuals in the sub–populations is replaced by fitter ones. The fitness progress for other configurations is very similar to the one in figure 5.

Table III presents the results of evolution when DOs are evolved against Random and GOJen (Configuration 2RG).

Table III. TEST GAME RESULTS (WIN PERCENTAGES) OF THE BEST DIGO PLAYERS EVOLVED AGAINST RANDOM AND GOJEN.

	Random	Naixt	GOJen			Strength
	5×5	5×5	5×5	7×7	9×9	5×5
Min	0.858	0.062	0.063	0.001	0.000	0.327
Avg	0.968	0.234	0.510	0.018	0.006	0.571
Max	0.997	0.428	0.823	0.103	0.104	0.705

Against GOJen, the maximum win percentage (0.823) and the maximum strength (0.705) out of the 20 diGOs are rather high, and the average win percentage of 51 percent seems to indicate that diGO has learned to play at the level of the best computer player used in evolution. However, looking at the Naixt column it becomes apparent that the success against GOJen is at least partly based on specialization to this opponent, even though, the win rates against Random are as high as should be expected.
A single diGO plays at a reasonable win rate of 0.103 against GOJen on the 7×7 board, which may partly be due to the increasing quality of GOJen on larger boards. However, when the board size is further increased to 9×9 the win rate of the best diGO remains at exactly the same level (0.104), which again, demonstrates the DO potential of board scalability.

C. Evolution against Random, Naixt, and GOJen

With configuration 3RNG all three computer opponents are used for fitness evaluation. The goal is to evolve players with general Go playing abilities that are not over–adapted to winning against a specific opponent.
The results of the 20 evolutionary runs can be seen in table IV.

Table IV. TEST GAME RESULTS (WIN PERCENTAGES) OF THE BEST DIGO PLAYERS EVOLVED AGAINST RANDOM, NAIXT, AND GOJEN.

	Random	Naixt			GOJen	Strength
	5×5	5×5	7×7	9×9	5×5	5×5
Min	0.899	0.267	0.117	0.011	0.098	0.451
Avg	0.966	0.393	0.208	0.085	0.330	0.563
Max	0.994	0.531	0.422	0.249	0.822	0.689

The strength values of the best diGOs are comparable to those with configuration 2RG (Table III), but the minimum strength is higher (0.451 over 0.327). The maximum win percentage against GOJen is very close to the one in the previous experiment (82.2%), although, the number of fitness games against GOJen has been reduced from 16 to 12 (Table I).
Looking at the performance of diGOs against the Naixt player, an improved scaling behavior can be observed, when compared to configuration 1N (Table II), where Naixt was the only opponent during evolution.
Specifically, the minimum win rate on 7×7 (0.117) indicates that in this configuration all diGOs are able to develop at least some general Go playing abilities on the 5×5 board, which are also applicable on the 7×7 board.

D. Game Analysis

In this section some selected games are discussed in order to closer investigate the capabilities of diGO players. It should be stressed that games have been chosen, where diGO wins against its computer opponent, which does not imply that it wins all or most games against this opponent, which is evident from the experimental results in this section. However, it should demonstrate that diGO is able to make meaningful moves, eventually, leading to a win.

1) Games against GOJen: The game shown in figure 6 shows a typical game of the diGO with strength 0.705 having emerged in run 19 of the experiment in configuration 2RG. It plays with the black stones against GOJen.

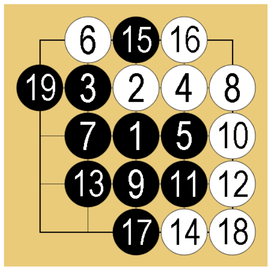
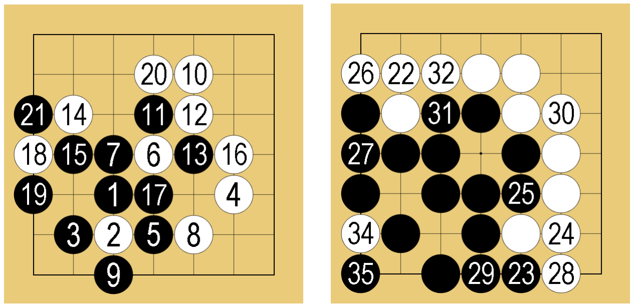


Figure 6. diGO (black) wins against GOJen (white) by 19.5 points.

diGO opens with the (known) optimal move on 5×5 in the center of the board. It then pushes the opponent into the top right corner and secures the territory in the bottom left corner. Though, some black moves are of low quality (e.g., 9, 13, 15), they do not reduce the overall positional advantage of black. In other words, after gaining a very good position in the first moves, black secures its advantage, and does not make a "silly" move, which is often a problem with evolved players. In the final board position the white group is dead and black wins by 19.5 points (komi 4.5).
A game of the strongest diGO player on the 7×7 board with a strength of 0.448 evolved in configuration 2RG is shown in figure 7. diGO plays the white stones versus GOJen on 7×7.

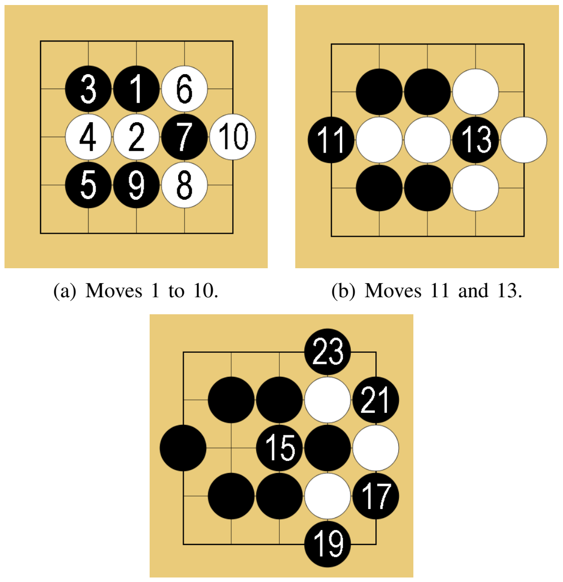


(a) Moves 1 to 21 (b) Moves 22 to 35. Move 33: pass.

Figure 7. diGO (white) wins against GOJen (black) by 9.5 points.

While GOJen is (too much) concerned capturing white stones (e.g., 2, 6, and 18), diGO plays some nice strategic moves (e.g., 4 and 10) marking territory at the sides and corners. Actually, this diGO has developed a very interesting (simple) strategy, which can also be observed on larger boards. It predominantly places stones on intersections, which are next to an intersection at the border (e.g., 10, 20, and 30). It even sacrifices stones in order to do so, which in the long run yields more territory, as can be seen nicely with the final board position in figure 7(b), when diGO wins the game by 9.5 points (komi 4.5).

2) Game against GNU Go: The best diGO player having emerged from configuration 2RG was able to win 34 percent of the games against GNU Go (level 6 out of 10) on 5×5. A selected game of this diGO against GNU Go is shown in figure 8 with GNU Go playing white.



(a) Moves 1 to 10. (b) Moves 11 and 13.

(c) Moves 15 to 23.

Figure 8. diGO (black) wins against GNU Go (white) by 13.5 points.

diGO opens at an intersection next to the center, which is known to lead to a sure win, if both sides play optimally [19]. GNU Go answers with the center move. Though, white captures 7, the black attack 11 on the white center stones 2 and 4 leads GNU Go to pass immediately, as it cannot save its stones.
The following black moves secure diGOs win by a margin of 13.5 (komi 4.5). Note that GNU Go is certainly not optimized for play on the 5×5 board, but this game shows that diGO, again, is able to play good moves against a tough computer opponent.

VII. Summary and Outlook

We have presented experiments evolving a digital organism (DO) to play the game of Go. More precisely, based on genetic programming we evolved a single cell program controlling each cell of the artificial organism, which "lives" on the intersections of the Go board. The cells communicate by the emission and absorption of mediators also regulated by the cell program. A cell's state indicates if a move should be made at the cell's intersection, or if the intersection should remain empty. Among other activities cells may replicate, grow, or die. These actions are, again, controlled by the cell program.
We consider the main advantages of this approach to be the distribution of game intelligence among the cells and the natural representation of the game by the cells, which leads to the potential for scalability, i.e., a DO having been evolved on a smaller board could be transferred to a larger board without any modifications. With Go this is very attractive as computational cost is often prohibitive for evolution of players on large boards. A scalable DO is comparable to the way humans learn the game, when they start to play on small boards and generalize essential game concepts to larger boards.
The results of the presented experiments demonstrate that DO evolution generates artificial players having the potential to win against a set of computer opponents on small boards (5×5 and

7×7). It could also be shown that the evolved DOs exhibit the assumed scalability in some cases, however, on average the win rates decreased on larger boards. This may partly be attributed to the fact that we evolved DOs on the 5×5 board, where essential concepts like corner play cannot be learned (as a corner on 5×5 is more or less the whole board).

There are many possible directions for future research. In order to improve the scalability DOs could be evolved on 7×7 or 9×9 boards, where game play closer resembles large board situations. In a further step board size may vary during evolution starting with small boards and increasing the board size in later stages. While these extensions are "only" a matter of computational power, the addition of game tree search to a DO player poses some problems, as all potential board situations have to be played out to assess the move quality. Hereby, the DO changes and is influenced by previous (bad) situations. It remains to be investigated how this affects the DO's quality of play.

Another idea is inspired by different embryonic stages presented in [20]. Here, different cell programs are evolved for different stages of DO development. E.g., for different game stages like opening, midgame, and endgame, specific cell programs may control the DO player. Similarly, the cell programs could differ in regions of the board, e.g., a corner, a side, and a center cell program.

Finally, the application of DOs to other (board) games leaves a broad spectrum of interesting research activities.

REFERENCES

[1] C. Chikun, *Go - A complete introduction to the game*. Tokyo, Amsterdam, Santa Monica: Kiseido Publishing Company, 1997.

[2] C.-S. Lee, M.-H. Wang, T.-P. Hong, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, and Y.-H. Kuo, "A novel ontology for computer Go knowledge management," in *Proceedings of the 18th international conference on Fuzzy Systems*, ser. FUZZ-IEEE'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1056–1061. [Online]. Available: http://dl.acm.org/citation.cfm?id=1717561.1717744

[3] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Learning To Evaluate Go Positions Via Temporal Difference Methods," Computational Intelligence in Games. Volume 62, Technical Report, 2000.

[4] F. A. Dahl, "Honte, a Go-Playing Program Using Neural Nets," in *In Workshop on Machine learning in Game Playing*. Nova Science Publishers, 1999, pp. 205–223.

[5] M. Enzenberger, "The Integration of A Priori Knowledge into a Go Playing Neural Network," University of Munich, Technical Report, 1996.

[6] H. A. Mayer, "Board Representations for Neural Go Players Learning by Temporal Difference," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2007)*. IEEE, 2007, pp. 183–188.

[7] N. Richards, D. Moriarty, and R. Miikkulainen, "Evolving Neural Networks To Play Go," in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97, East Lansing, MI)*, T. Bäck, Ed. San Francisco, CA: Morgan Kaufmann, 1998, pp. 768–775. [Online]. Available: http://nn.cs.utexas.edu/?richards:apin97

[8] A. Lubberts and R. Miikkulainen, "Co-Evolving A Go-Playing Neural Network," in *Coevolution: Turning Adaptive Algorithms Upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, p. 6. [Online]. Available: http://nn.cs.utexas.edu/?lubberts:geccoworkshop01

[9] H. A. Mayer and P. Maier, "Coevolution of Neural Go Players in a Cultural Environment," in *Proceedings of the Congress on Evolutionary Computation 2005*. IEEE Press, 2005.

[10] J. R. Koza, "Genetic Programming as a Means for Programming Computers by Natural Selection," *Statistics and Computing (UK)*, vol. 4, pp. 191–198, 1994.

[11] ——, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[12] D. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[13] J. F. Miller, "Evolving a self-repairing, self-regulating, french flag organism," in *Genetic And Evolutionary Computation Conference (GECCO)*. Springer Verlag, 2004, pp. 129–139.

[14] D. Roggen and D. Federici, "Multi-Cellular Development: Is There Scalability and Robustness to Gain?" in *Proceedings of Parallel Problem Solving from Nature 8, Parallel Problem Solving from Nature (PPSN) 2004*. Springer Verlag, 2004, pp. 391–400.

[15] T. G. W. Gordon and P. J. Bentley, "Bias and scalability in evolutionary development," in *In Genetic And Evolutionary Computation Conference (GECCO) '05*. ACM Press, 2005, pp. 83–90.

[16] L. Bai, M. Eyiyurekli, and D. E. Breen, "Automated shape composition based on cell biology and distributed genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2008.

[17] T. Kowaliw, P. Grogono, and N. Kharma, "Environment as a spatial constraint on the growth of structural form," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ser. Genetic And Evolutionary Computation Conference (GECCO) '07. New York, NY, USA: ACM, 2007, pp. 1037–1044. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277163

[18] D. Whitley, S. Rana, and R. B. Heckendorn, "The Island Model Genetic Algorithm: On Separability, Population Size and Convergence," *Journal of Computing and Information Technology*, vol. 7, pp. 33–47, 1998.

[19] E. C. D. van der Werf, H. J. V. D. Herik, and J. W. H. M. Uiterwijk, "Solving Go On Small Boards," *International Computer Games Association Journal*, vol. 26, pp. 10–7, 2003.

[20] D. Federici, "Using Embryonic Stages to increase the evolvability of development," in *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004) Workshop Program*. Springer Verlag, 2004.