Constituent Grammatical Evolution

Loukas Georgiou and William J. Teahan
School of Computer Science, Bangor University
Bangor, Wales
loukas@informatics.bangor.ac.uk and w.j.teahan@bangor.ac.uk

Abstract

We present Constituent Grammatical Evolution (CGE), a new evolutionary automatic programming algorithm that extends the standard Grammatical Evolution algorithm by incorporating the concepts of constituent genes and conditional behaviour-switching. CGE builds from elementary and more complex building blocks a control program which dictates the behaviour of an agent and it is applicable to the class of problems where the subject of search is the behaviour of an agent in a given environment. It takes advantage of the powerful Grammatical Evolution feature of using a BNF grammar definition as a plug-in component to describe the output language to be produced by the system. The main benchmark problem in which CGE is evaluated is the Santa Fe Trail problem using a BNF grammar definition which defines a search space semantically equivalent with that of the original definition of the problem by Koza. Furthermore, CGE is evaluated on two additional problems, the Loss Altos Hills and the Hampton Court Maze. The experimental results demonstrate that Constituent Grammatical Evolution outperforms the standard Grammatical Evolution algorithm in these problems, in terms of both efficiency (percent of solutions found) and effectiveness (number of required steps of solutions found).

1 Background

1.1 Grammatical Evolution

Grammatical Evolution [O'Neill and Ryan, 2003] is an evolutionary algorithm that can evolve complete programs in an arbitrary language using populations of variable-length binary strings. Namely, a chosen evolutionary algorithm (typically a variable-length genetic algorithm) creates and evolves a population of individuals and the binary string (genome) of each individual determines which production rules in a Backus Naur Form (BNF) grammar definition are used in a genotype-to-phenotype mapping process to generate a program. According to O'Neill and Ryan [2001], Grammatical Evolution's unique features compared to other evolutionary algorithms are the degenerate genetic code which facilitates the occurrence of neutral mutations (various genotypes can represent the same phenotype), and the wrapping of the genotype during the mapping process which enables the reuse of the same genotype for the production of different phenotypes.

Grammatical Evolution (GE) takes inspiration from nature. It embraces the developmental approach and draws upon principles that allow an abstract representation of a program to be evolved. This abstraction enables the following: the separation of the search and solution spaces; the evolution of programs in an arbitrary language; the existence of degenerate genetic code; and the wrapping that allows the reuse of the genetic material.

Before the evaluation of each individual, the following steps take place in Grammatical Evolution:

i. The genotype (a variable-length binary string) is used to map the start symbol of the BNF grammar definition into terminals. The grammar is used to specify the legal phenotypes.

ii. The GE algorithm reads "codons" of 8 bits and the integer corresponding to the codon bit sequence is used to determine which form of a rule is chosen each time a non-terminal to be translated has alternative forms.

iii. The form of the production rule is calculated using the formula *form = codon mod forms* where *codon* is the codon integer value, and *forms* is the number of alternative forms for the current non-terminal.

Grammatical Evolution has been shown to be an effective and efficient evolutionary algorithm in a series of both static and dynamic problems [Dempsey *et al.*, 2009]. However, there are known issues that still need to be tackled such as destructive crossovers [O'Neill *et al.*, 2003], genotype bloating [Harper and Blair, 2006], and low locality of genotype-to-phenotype mapping [Rothlauf and Oetzel, 2006].

1.2 The Santa Fe Trail Problem

The Santa Fe Trail (SFT) is an instance of the Artificial Ant problem and it is standard problem used for benchmarking purposes in the areas of Genetic Programming [Koza, 1992; Koza, 1994] and Grammatical Evolution [O'Neill and Ryan, 2001; O'Neill & Ryan, 2003]. Its objective is to find a computer program to control an artificial ant, so that it can find all 89 pieces of food located on the grid.

The more general Artificial Ant problem was devised by Jefferson *et al.* [Koza, 1992, p.54] with the Genesys-Tracker System [Jefferson *et al.*, 1992]. The Santa Fe Trail was designed by Christopher Langton [Koza, 1992, p.54], and is a somewhat more difficult trail than the “John Muir Trail” originally used in the Artificial Ant problem. Although there are other more difficult trails than the Santa Fe Trail, such as the “Los Altos Hills” [Koza, 1992, p. 156], it is acknowledged as a challenging problem for evolutionary algorithms to tackle. The Santa Fe Trail has the following irregularities [Koza, 1992, p.54]: single gaps, double gaps, single gaps at corners, double gaps at corners (short knight moves), and triple gaps at corners (long knight moves).

The artificial ant of the Santa Fe Trail problem operates in a square 32x32 toroidal grid in the plane. The ant starts in the upper left cell of the grid (0, 0) facing east. The trail consists of 144 squares with 21 turns, and the 89 units of food are distributed non-uniformly along it. The artificial ant can use three operations: *move*, *turn-right*, *turn-left*. Each of these takes one time unit. In addition, the artificial ant can use one sensing function: *food-ahead*. It looks into the square the ant is currently facing and returns true or false depending upon whether that square contains food or is empty respectively.

It has been shown by Langdon and Poli [Robilliard *et al.*, 2006] that Genetic Programming does not improve much on pure random search in the Santa Fe Trail. This problem has become quite popular as a benchmark in the GP field and is still repeatedly used [Robilliard *et al.*, 2006]. Hugosson *et al.* [2010] mention that the Santa Fe Trail has the features often suggested in real program spaces – it is full of local optima and has many plateaus. Furthermore, they note that a limited GP schema analysis shows that it is deceptive at all levels and that there are no beneficial building blocks, leading Genetic Algorithms to choose between them randomly. These reasons make the Santa Fe Trail a challenging problem for Evolutionary Algorithms.

1.3 Performance of GE on the Santa Fe Trail

Robilliard *et al.* [2006] say that in order to evaluate GP algorithms using the Santa Fe Trail problem as a benchmark “… the search space of programs must be semantically equivalent to the set of programs possible within the original SFT definition”. However, it has been argued that Koza’s [Koza, 1992] and GE’s [O'Neill and Ryan, 2003] search spaces are not semantically equivalent [Robilliard *et al.*, 2006].

Georgiou and Teahan [2010] have proved experimentally the above claim and furthermore show that Grammatical Evolution gives very poor results in the Santa Fe Trail problem when a BNF grammar is used which defines a search space semantically equivalent with the search space used in the original problem [Koza, 1992]. Indeed, the same work proved experimentally that GE literature [O’Neill and Ryan, 2001; O’Neill and Ryan, 2003] uses a BNF grammar which biases the search space and consequently gives an unfair advantage in GE when it is compared against Genetic Programming in the Santa Fe Trail problem. Furthermore, it has been shown that during an evolutionary run, GE is not capable of finding solutions using the BNF grammar definition it uses in its benchmark against GP with less than 607 steps.

When biasing the search space with the incorporation of domain knowledge (namely, suiting a grammar to the problem in question), this is a great advantage not only of Grammatical Evolution but of any grammar-based Evolutionary Algorithm. But this is irrelevant when comparing the actual performance of an Evolutionary Algorithm because in the case of GE experiments mentioned in the literature, it is the human designed bias of the search space that improves the success rate of the algorithm. Consequently, it is not fair to compare GE with GP when semantically different search spaces are used. However, if the bias could emerge or be enforced during the evolution through some other mechanism (and not by design) this would be a different case; for example, through evolution of the grammar or a different mechanism inspired by nature and biology like the one proposed in this work.

2 Constituent Grammatical Evolution

2.1 Motivation

The goal of Constituent Grammatical Evolution (CGE) is to improve Grammatical Evolution in terms of effectiveness (percent of solutions found in a total of evolutionary runs) and efficiency (solution quality in terms of the characteristics of the problem in question) in agent problems like the Artificial Ant [Koza, 1992]. The main benchmark problem in which CGE is first evaluated is the Santa Fe Trail problem using a BNF grammar definition which defines a search space semantically equivalent with this of the original definition of the problem by Koza [1992]. Consequently, the specific goals are: first, to improve the success rate of evolutionary runs in the Santa Fe Trail problem against standard GE; and second, to find solutions which will require fewer steps than the solutions the GE algorithm is usually able to find.

Constituent Grammatical Evolution takes inspiration from two very elementary concepts: genes and conditional behaviour switching. It augments the standard Grammatical Evolution algorithm by incorporating and utilising these concepts in order to improve the performance of the later in agent problems.

The proposed evolutionary algorithm constitutes a conditional behaviour switching evolutionary algorithm, based on the Grammatical Evolution algorithm, and which incorporates the notion of genes in the individual’s genotype. CGE builds from elementary and more complex building blocks a control program which dictates the behaviour of an agent. It is applicable to the class of problems where the subject of search is the behaviour of an agent in a given environment, like the Santa Fe Trail problem. CGE’s current implementation takes advantage of the powerful Grammatical Evolution feature of using a BNF grammar definition as a plug-in component to describe the output language to be produced by the system (control program). Namely, it utilizes this powerful BNF based phenotype-to-genotype feature of GE which enables the implementation of the CGE's unique features (genes and conditional behaviour switching) in a straightforward and very flexible way.

The main GE issues tackled by CGE are the following:

• decreasing the actual search space without using domain knowledge of the problem in question and without changing the semantics of the search space defined by the BNF grammar compared with the original problem;

• preventing destructive crossover events of Grammatical Evolution; and

• reducing the bloating of the genotype of the Grammatical Evolution algorithm.

The Constituent Grammatical Evolution algorithm augments the standard Grammatical Evolution algorithm, in order to tackle the above issues, with the incorporation of three unique features:

• the conditional behaviour switching approach, which biases the search space toward useful areas;

• incorporation of the notion of genes, which tackles the issue of destructive crossovers and provides useful reusable building blocks; and

• restriction of the genotype size, which tackles the phenotype bloat phenomenon.

2.2 CGE Algorithm Description

The Constituent Grammatical Evolution algorithm uses the following inputs:

1. Problem Specification (*PS*): A program which simulates the problem in question and assigns to each individual and constituent gene a fitness value.

2. Language Specification (*BNF-LS*): A BNF grammar definition which dictates the grammar of the output programs of the individuals which are executed by the problem specification simulator.

3. Behaviour Switching Specification (*BNF-BS*): The BNF grammar definition which augments the Language Specification by incorporating in the original specification (*BNF-LS*) the switching behaviour approach. Namely, the *BNF-BS* specification is a BNF grammar definition which enforces a conditional check before each agent's action and which results (in conjunction with the phenotypes of the constituent genes) to the BNF grammar definition used for the evolution of the agents. Using such an approach in the Artificial Ant problem, for example, the resultant ant control programs must perform a check (condition) whether there is food ahead or not. In this way the created control programs can be expressed as simple finite-state machines and a form of memory is incorporated indirectly in the ant's behaviour.

4. Grammatical Evolution Algorithm (*GE*): The genetic algorithm which is used for the evolution of the population of the individuals.

The overall description of the Constituent Grammatical Evolution algorithm (Listing 1) is as follows.

Using the CGE specific input parameters (*IMC*, *S*, *G*, *L*, *E*, *CMin*, *CMax*, and *W*) as defined below, a pool *P* of constituent genes is created and filled in the following way. *S* genes are randomly created which are represented as binary strings. These genes will be candidates for the pool of the constituent genes. The minimum length of each string is *CMin* codons and the maximum length is *CMax* codons (the size in bits of each codon is specified by the GE algorithm; usually its value in the GE literature is eight).

```
Set current constituent genes pool P of size S
Repeat G times
  Create a new empty pool P′
  Repeat until P′ is full
    Create gene N with genotype size L codons
    Set i = 0
    While i < E
      Map genotype of N to its phenotype (wrap W)
      Put N in a random environment location
      Set direction of N randomly
      Set interim fitness value Vi of N to 0
      Repeat L times
        Execute phenotype (program) of N
        Calculate performance Vi′ of N
        Set Vi = Vi + Vi′
      End Repeat
    End While
    Calculate and set fitness value F of gene N
    Add N to P′
  End Repeat
  Create a pool T of size 2*S
  Add to T the genes of P and P′
  Sort T according the fitness values F of genes
  Create an empty P′′
  Add to P′′ the best S genes of T
  Replace P with P′′
End Repeat
For each gene N in P
  Add phenotype of N in BNF-BS grammar definition
End For
Create population R of individuals
Evolve population R enforcing IMC limit to indi-
vidual′s genotype
```

Listing 1: The Constituent Grammatical Evolution algorithm.

The fitness function for the evaluation of the candidate constituent genes depends on the problem specification and is similar to the fitness function used for the individuals in the problem in question. The general concept behind the creation and evaluation of the constituent genes is to find reusable modules from which the genotype of the individual will be constructed. For demonstration purposes, the way constituent genes are evaluated (described below) is based on the implementation in the SFT problem.

The fitness value *F* of each gene is calculated as follows. The gene is placed in a random location of the environment with a random heading. Then the gene's genotype is mapped to its phenotype using the language specification *BNF-LS* (BNF grammar definition) and the Grammatical Evolution mapping formula with *W* genotype wraps limit. The gene's code (the phenotype as for an individual) is executed for L times and evaluated to produce an interim fitness value. The interim fitness value is calculated E times. Then, the final fitness value of the candidate constituent gene is calculated with the formula

$$F = ((V_1 + \ldots + V_N)/N) \times (V_{\max}/S) \quad (1)$$

where: F is the final fitness value of the constituent gene used for comparison with other genes and for deciding whether it will placed in the genes pool or not; N ($N \leq E$) is the number or interim evaluations with fitness value > 0; V_n is the Interim fitness value of the interim evaluation n ($0 \leq n \leq N$); V_{max} is the maximum interim fitness value found; and S is the number of steps performed during this interim evaluation.

The genes of each generation are compared with the genes of the previous generation according to their fitness value and the best S genes of these two generations are placed in the pool P replacing the existing genes. This will be repeated for G generations. Finally, the pool P will be filled with the best S genes (and their corresponding phenotypes) created during the G generations.

When the genes pool is finally created and filled, the phenotypes of the constituent genes are added as terminal operators in the behaviour switching BNF grammar definition (*BNF-BS*). Then an initial population of individuals is created randomly where each individual consists of a control gene which dictates which original terminal operators and which constituent genes of the *BNF-BS* grammar definition are used and in what order. Finally, the population of individuals is evolved using the standard Grammatical Evolution algorithm, enforcing the individual's genotype size maximum limit *IMC* after each crossover.

3 Experimental Results

3.1 The Santa Fe Trail Problem

The performance of the Constituent Grammatical Evolution (CGE) algorithm has been benchmarked against the Grammatical Evolution (GE) algorithm using the Santa Fe Trail problem. Two distinct benchmarks have been performed for the following situations: a) when GE uses the standard BNF grammar definition mentioned in the Grammatical Evolution literature [O'Neill and Ryan, 2001; O'Neill and Ryan 2003] for the SFT problem, which defines a search space biased and not semantically equivalent with that of the original problem [Koza, 1992], named here BNF-O'Neill; and b) a BNF grammar named BNF-Koza which defines a search space semantically equivalent with the search space of the original problem as it was defined by Koza [1992]. When the Grammatical Evolution (GE) algorithm uses the BNF-O'Neill grammar, we will refer to it as "GE using BNF-O'Neill". In the same way, when it uses the BNF-Koza grammar, we will refer to it as "GE using BNF-Koza".

For these benchmarks, three series of experiments were performed to evaluate CGE, GE using BNF-Koza, and GE using BNF-O'Neill. In each series of experiments, five distinct experiments were conducted consisting of one hundred evolutionary runs each. Namely, a total of five hundred evolutionary runs were performed for each algorithm. The tableau in Table 2 shows the GE settings and parameters of each evolutionary run and the Table 3 shows the CGE specific parameters. The BNF grammars used in these experiments are BNF-BS (Listing 4), BNF-Koza, and BNF-O'Neill [Georgiou and Teahan, 2010].

Objective	Find a computer program in the NetLogo programming language to control an artificial ant so that it can find all 89 pieces of food located on the Santa Fe Trail.
Terminal Operators	*turn-left, turn-right, move, food-ahead*, plus constituent genes when the CGE algorithm is used instead of the standard GE algorithm.
Raw Fitness	Number of pieces of food picked up before the ant times out with 650 operations
BNF Grammar	CGE: BNF-Koza (original) for Genes and BNF-BS (behaviour switching) for Ants. GE using BNF-Koza: BNF-Koza. GE using BNF-O'Neill: BNF-O'Neill.
Evolutionary Algorithm	Steady-State Genetic Algorithm, Generation Gap = 0.9 Selection Mechanism: Roulette-Wheel Selection
Initial Population	Randomly created with the following restrictions: Minimum Codons = 15 and Maximum Codons = 25
Parameters	Population Size = 500, Maximum Generations = 50 Mutation Prob. = 0.01, Crossover Prob. = 0.9 Codon Size = 8, Wraps Limit = 10

Table 2: GE Tableau for the Santa Fe Trail.

Codons Limit, *IMC*	250	Gene Evaluations, *E*	50
Gene Pool Size, *S*	3	Gene Codons Min, *CMin*	10
Gene Generations, *G*	850	Gene Codons Max, *CMax*	20
Gene Code Iterations, *L*	10	Gene Max Wraps, *W*	5

Table 3: CGE Settings for the Santa Fe Trail.

```
N = {behaviour, op}
T = {turn-left, turn-right, move, ifelse,
     food-ahead, [, ]}
S = behaviour
P:
<behaviour> ::= ifelse food-ahead [ <op> ]
                [ <op> <behaviour> ]
            | ifelse food-ahead [ <op> ][ <op> ]
            | <op>
<op> ::= turn-left | turn-right | move
       | ... {Constituent Genes Phenotypes}(*)
```

Listing 4: The BNF-BS Grammar Definition for the Santa Fe Trail problem. The phenotype of every constituent gene (*) is added as a production rule in the <op> non-terminal symbol. BNF-BS defines a search space semantically equivalent with that of the original problem [Koza, 1992].

Constituent Grammatical Evolution was successful at finding a solution in the Santa Fe Trail problem with very high success rates, ranging from 85% to 94% with an average success rate of 90%. The best solution found by CGE, requires only 337 steps and there is no GP or GE publication in our knowledge presenting a solution requiring less steps. The NetLogo code for this solution is shown in Listing 5.

```
ifelse food-ahead
  [move]
  [turn-right
   ifelse food-ahead
     [ifelse food-ahead
       [ifelse food-ahead
         [move move] [move]
        ifelse food-ahead
          [move move] [move]
       ]
       [turn-left]
      move
     ]
     [ifelse food-ahead
        [move] [turn-right]
      ifelse food-ahead
        [turn-left]
        [turn-right
         ifelse food-ahead
           [move]
           [ifelse food-ahead
             [move] [turn-right]
            move
  ]  ]  ]  ]
```

Listing 5: NetLogo code of the best solution (in terms of required steps) found by CGE in the Santa Fe Trail problem.

The Table 6 shows the detailed results of the experiments. The column "Best" shows the best value of all five experiments. "Runs" is the number of evolutionary runs performed in the experiment, "Steps", the required steps of the best solution found in the particular experiment, "Success", how many evolutionary runs (percentage) found a solution, and "Avg.Suc.", the average success rate of all five experiments.

	Exp #1	Exp #2	Exp #3	Exp #4	Exp #5	**Best**
Runs	100	100	100	100	100	**100**
Steps	393	375	393	377	337	**337**
Success	85%	93%	89%	94%	87%	**94%**
Avg.Suc.	**90%**					

Table 6: CGE Experimental Results in SFT.

	Exp #1	Exp #2	Exp #3	Exp #4	Exp #5	**Best**
Runs	100	100	100	100	100	**100**
Steps	419	507	415	541	479	**415**
Success	8%	11%	10%	6%	13%	**13%**
Avg.Suc.	**10%**					

Table 7: GE using BNF-Koza Experimental Results in SFT.

	Exp #1	Exp #2	Exp #3	Exp #4	Exp #5	**Best**
Runs	100	100	100	100	100	**100**
Steps	609	609	607	609	607	**607**
Success	80%	76%	75%	81%	74%	**81%**
Avg.Suc.	**78%**					

Table 8: GE using BNF-O'Neill Experimental Results in SFT.

The results of the experiments using the standard GE with BNF-Koza and BNF-O'Neill can be seen in Table 7 and in Table 8. A cumulative frequency measure of success over 500 runs of CGE and GE can be seen in Figure 9.

The experimental results show that Constituent Grammatical Evolution outperforms Grammatical Evolution in terms of success rate whether the later uses a BNF grammar definition (BNF-Koza) which defines a search space semantically equivalent with that used in the original problem [Koza, 1992], or whether it uses a BNF grammar definition (BNF-O'Neill) which defines a biased search space. In contrast, Constituent Grammatical Evolution is able to find much better solutions in terms of the required steps.

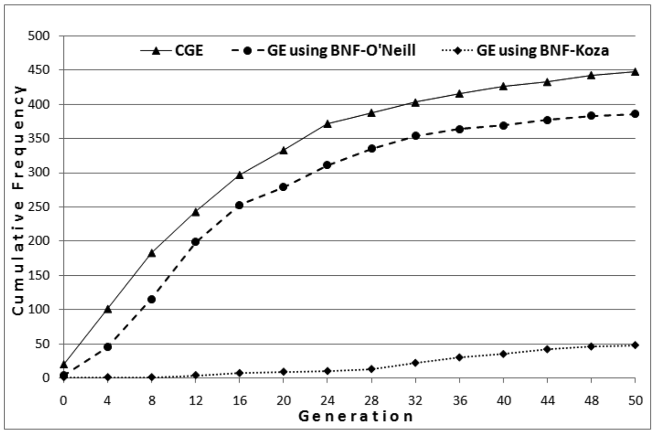

Figure 9: CGE vs. GE on the Santa Fe Trail problem.

The poor performance of GE using BNF-Koza can be explained if we take in account that this grammar definition defines a search space semantically equivalent with that of the original problem, which is very large, making it difficult for GE to find a solution due to its main issues which were mentioned in section 1.1. Instead, BNF-O'Neill defines a smaller search space and biases the search to areas where a solution can be found more easily (with higher success rate) but with the cost of excluding other areas where more efficient programs could be found (using less steps). This is the reason why GE using BNF-O'Neill didn't find solutions with less than 607 steps. In order to support the last argument, a series of five new experiments of one hundred evolutionary runs each was conducted, setting the steps limit to 606. In these experiments the success rate of GE was 0%, providing further experimental evidence that GE using BNF-O'Neill cannot find solutions with less than 607 steps.

Furthermore, Georgiou & Teahan [2010] tried to solve the Santa Fe Trail problem using random search as the search mechanism of Grammatical Evolution instead of the standard steady-state genetic algorithm. Their results show that using the search space defined by the BNF-O'Neill grammar definition, GE with random search has a success rate of 50% in finding a solution. Instead, the success rate of GE with random search using the search space defined by the BNF-Koza grammar definition is just 1.4%. These results further support the claim that the BNF-O'Neill grammar definition defines a different search space than the

BNF-Koza grammar definition, where solutions can be found more easily.

The promising results of CGE benchmarking on the Santa Fe Trail problem raises the question whether it can outperform GE in other problems as well. For this reason, CGE was applied and benchmarked on two additional problems. The first is a more difficult version of the Santa Fe Trail problem and the second a typical maze search problem.

3.2 The Los Altos Hills Problem

The Los Altos Hills (LAH) problem is an instance of the Artificial Ant problem and was introduced by Koza [1992, p. 156]. The objective of this problem is to find a computer program to control an artificial ant, so that it can find all 157 pieces of food located on a 100x100 toroidal grid. The ant starts in the upper left cell of the grid facing east. The trail consists of 221 squares with 29 turns. The artificial ant can uses the same operations as with the Santa Fe Trail problem.

The Los Altos Hills trail is larger and more difficult than the Santa Fe Trail. It begins with the same irregularities and in the same order. Then it introduces two new kinds of irregularity that appear toward the end of the trail.

In order to evaluate CGE against GE, a series of three experiments was conducted (CGE, GE using BNF-Koza, and GE using BNF-O'Neill), consisting of 100 evolutionary runs each using the configuration shown in Table 10 and in Table 11. The constituent genes evaluation formula used in this problem was the same as that used on the Santa Fe Trail.

The BNF-BS grammar definition used in this problem is shown in Listing 12. Note that a slightly different BNF-BS grammar definition is used than that used for the Santa Fe Trail problem. These modifications in the BNF-BS grammar definition were applied because the solution to the Los Altos Hills problem (according to its specifications) requires less efficient individuals in terms of the fraction of steps and food pieces. Namely, the fraction is 19.2 (3000/156) against 7.3 (650/89) for the Santa Fe Trail.

Objective	Find a computer program in the NetLogo programming language to control an artificial ant so that it can find all 157 pieces of food located on the Los Altos Hills trail.
Terminal Operators	*turn-left, turn-right, move, food-ahead*, plus constituent genes when the CGE algorithm is used instead of the standard GE algorithm.
Raw Fitness	Number of pieces of food picked up before the ant times out with 3000 operations
BNF Grammar	CGE: BNF-Koza (original) for Genes and BNF-BS (behaviour switching) for Ants. GE using BNF-Koza: BNF-Koza. GE using BNF-O'Neill: BNF-O'Neill.
Evolutionary Algorithm	Steady-State Genetic Algorithm, Generation Gap = 0.9 Selection Mechanism: Roulette-Wheel Selection
Initial Population	Randomly created with the following restrictions: Minimum Codons = 50 and Maximum Codons = 100
Parameters	Population Size = 2000, Maximum Generations = 50 Mutation Prob. = 0.01, Crossover Prob. = 0.9 Codon Size = 8, Wraps Limit = 10

Table 10: GE Tableau for the Los Altos Hills problem.

Codons Limit, *IMC*	750	Gene Evaluations, *E*	100
Gene Pool Size, *S*	3	Gene Codons Min, *CMin*	40
Gene Generations, *G*	1000	Gene Codons Max, *CMax*	100
Gene Code Iterations, *L*	10	Gene Max Wraps, *W*	10

Table 11: CGE Settings for the Los Altos Hills problem.

```
N = {behaviour, op}
T = {turn-left, turn-right, move, ifelse,
     food-ahead, [, ]}
S = behaviour
P:
<behaviour> ::=
 <behaviour> ifelse food-ahead [<op>][<op> <behaviour>] |
 <behaviour> ifelse food-ahead [<op>][<op> <behaviour>] |
 ifelse food-ahead [ <op> ][ <op> <behaviour> ] |
 ifelse food-ahead [ <op> ][ <op> <behaviour> ] |
 ifelse food-ahead [ <op> ][ <op> ] |
<op><op> ::= turn-left | turn-right | move
        | ... {Constituent Genes Phenotypes}
```

Listing 12: The BNF-BS Grammar Definition for LAH.

	CGE	GE BNF-Koza	GE BNF-O'Neill
Runs	100	100	100
Best Solution's Steps	1093	No solution	No solution
Success Rate	9%	0%	0%

Table 13: Experimental Results of the Los Altos Hills problem.

The experimental results are shown in Table 13. Even though the Los Altos Hill is a much more challenging problem than the Santa Fe Trail, CGE managed to find a solution in contrast to GE which wasn't able to find one.

The main reason why LAH proved to be so difficult for GE is: first, the two new irregularities introduced which require a more complex behaviour by the ant; and second, because these irregularities first appear at the end of the trail, with the consequence that the evolved population converges to programs tackling only the first part of the trail which is identical to the Santa Fe Trail.

3.3 The Hampton Court Maze Problem

The third problem where CGE was benchmarked against GE is a typical maze searching problem [Teahan, 2010a]. The maze used for the experiments is the Hampton Court Maze [Teahan, 2010b; p. 79] which is a simple connected maze of grid size 39x23 (Figure 14). The goal is to find a program guiding the traveller agent from the entry square of the maze to the centre of the maze.

As with the LAH, three experiments were conducted of one hundred evolutionary runs each in order to compare the performance of CGE, GE using BNF-Koza, and GE using BNF-O'Neill. The BNF grammar definitions used in these experiments are variations of those used in the SFT problem with the following differences: the replacement of the *food-ahead* sensing operator with the *wall-ahead?* operator, the addition of two sensing operators, *wall-left?* and *wall-right?*; and the addition of a non-terminal symbol for choosing any one of these sensing operators when a condition statement is to be selected. The sensing operators were inspired by the work of Sondahl [2005].

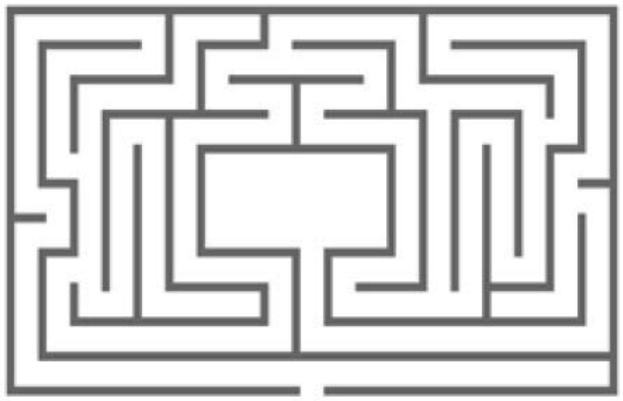

Figure 14: The Hampton Court Maze. Reprinted from [Teahan, 2010b; p. 79].

The experimental configuration is shown in Tables 15 and 16. The fitness value of the candidate constituent genes is calculated with the formula

$$F = ((V_1 + \ldots + V_E)/E) \times (V_{\max}/S) \tag{2}$$

where: F is the fitness value of the constituent gene used for comparison with other genes and for deciding whether it will placed in the genes pool or not; E is the number of performed gene's interim evaluations; V_n is the Interim fitness value of the interim evaluation n ($0 \le n \le E$); V_{max} is the maximum interim fitness value found; and S is the number of operations (time steps) performed during the interim evaluation of the gene with the maximum interim fitness.

Each interim fitness value of the candidate constituent genes is calculated with the formula

$$V = D_S - (D_P/P) \tag{3}$$

where: V is the interim fitness value; D_S is the geometric distance between the gene and the exit of the maze before the gene executes its code; D_P is the geometric distance between the gene and the exit of the maze after the execution of the gene's code for L times (Gene Code Iterations); and P is the number of new path squares visited during the gene's code execution.

The results of these experiments are shown in Table 17. The Hampton Court Maze problem proved to be difficult for GE, mainly because the first square the agent visits when it enters the maze is assigned a high fitness value due its small geometric distance from the exit. This leads to convergence of the population to this local optimum. Namely it advances individuals which just execute the *move* operator. In contrast, CGE proved to be very effective for this problem because it was able to easily overcome this local optimum due to the constituent genes. Indeed, the solution it found requires much fewer steps than those found by GE.

Objective	Find a computer program in the NetLogo programming language to control an artificial traveller agent so that it can find the centre of the maze.
Terminal Operators	*turn-left, turn-right, move, wall-ahead?, wall-left?, wall-right?*, plus constituent genes when the CGE algorithm is used instead of the standard GE algorithm.
Raw Fitness	The geometric distance between the agent's position and the entrance to the centre of the maze before the agent times out with 500 operations, divided by the number of new squares of the path visited.
Adjusted Fitness	1 / (1 + Raw Fitness)
BNF Grammar	CGE: BNF-Koza (maze version) for Genes and BNF-BS (behaviour switching maze version) for travellers (agents). GE using BNF-Koza: BNF-Koza (maze version). GE using BNF-O'Neill: BNF-O'Neill (maze version).
Evolutionary Algorithm	Steady-State Genetic Algorithm, Generation Gap = 0.9 Selection Mechanism: Roulette-Wheel Selection
Initial Population	Randomly created with the following restrictions: Minimum Codons = 15 and Maximum Codons = 25
Parameters	Population Size = 100, Maximum Generations = 25 Mutation Prob. = 0.01, Crossover Prob. = 0.9 Codon Size = 8, Wraps Limit = 10

Table 15: GE Tableau for the Hampton Court Maze problem.

Codons Limit, *IMC*	250	Gene Evaluations, *E*	50
Gene Pool Size, *S*	3	Gene Codons Min, *CMin*	10
Gene Generations, *G*	500	Gene Codons Max, *CMax*	20
Gene Code Iterations, *L*	10	Gene Max Wraps, *W*	5

Table 16: CGE settings for the Hampton Court Maze problem.

	CGE	GE BNF-Koza	GE BNF-O'Neill
Runs	100	100	100
Best Solution's Steps	384	439	494
Success Rate	82%	1%	1%

Table 17: Experimental results for the Hampton Court Maze problem.

4 Conclusions

Constituent Grammatical Evolution is a new evolutionary automatic programming algorithm that extends Grammatical Evolution by incorporating in the original algorithm the concepts of constituent genes and conditional behaviour-switching. CGE builds from elementary and more complex building blocks a control program which dictates the behaviour of an agent and it is applicable to the class of problems where the subject of search is the conditional behaviour of an agent in a given environment. It takes advantage of the powerful Grammatical Evolution feature of using a BNF grammar definition as a plug-in component to describe the output language to be produced by the system.

CGE is able through its conditional behaviour-switching feature to focus the search in more useful areas by decreasing the actual search space without using domain knowledge of the problem in question and without changing semantically the original search space. Indeed, due to the constituent genes and genotype size limit features of CGE, the Grammatical Evolution issues of destructive crossover events and genotype bloating are tackled respectively.

The experimental results presented in this paper show that CGE outperforms the standard GE algorithm in the Santa Fe Trail problem whether the later uses a search space semantically equivalent with that of the original GP problem [Koza, 1992], or whether it uses the biased search space of the standard Grammatical Evolution system (BNF-O'Neill) used in the Grammatical Evolution literature as was first presented by O'Neill and Ryan [2003]. Namely, CGE achieves better results than the standard Grammatical evolution algorithm in terms of both efficiency (percent of solutions found) and effectiveness (number of required steps of solutions found). Further experimental results show that CGE outperforms GE in two additional and more difficult problems, the Los Altos Hills and the Hampton Court Maze.

All experiments mentioned in this study have been performed using the jGE library [Georgiou and Teahan, 2006a; 2006b; 2008], which is a Java implementation of the Grammatical Evolution algorithm, and the jGE NetLogo extension, which is a NetLogo extension of the jGE library. The simulation of the Santa Fe Trail problem is implemented in the NetLogo modelling environment. The source code of the jGE library, the jGE NetLogo extension, and the Santa Fe Trail simulation in NetLogo, are freely available [Georgiou, 2006; Wilensky, 1999].

References

[Dempsey *et al*, 2009] Dempsey, I., O'Neill, M. and Brabazon, A. *Foundations in Grammatical Evolution for Dynamic Environments*. Berlin, Germany: Springer, 2009.

[Georgiou, 2006] Georgiou, L. *Java GE (jGE) Official Web Site*. School of Computer Science, Bangor University, Available from http://www.bangor.ac.uk/~eep201/jge, 2006.

[Georgiou and Teahan, 2006a] Georgiou, L. and Teahan, W. J. jGE – A Java implementation of Grammatical Evolution. *10th WSEAS International Conference on Systems*, Athens, Greece, July 10-15, 2006.

[Georgiou and Teahan, 2006b] Georgiou, L. and Teahan, W. J. Implication of Prior Knowledge and Population Thinking in Grammatical Evolution: Toward a Knowledge Sharing Architecture. *WSEAS Transactions on Systems*, 5(10), 2338-2345, 2006.

[Georgiou and Teahan, 2008] Georgiou, L. and Teahan, W. J. Experiments with Grammatical Evolution in Java. *Studies in Computational Intelligence*, 102, 45-62, 2008.

[Georgiou and Teahan, 2010] Georgiou, L. and Teahan, W. J. (2010) Grammatical Evolution and the Santa Fe Trail Problem. In *Proceedings of the International Conference on Evolutionary Computation (ICEC 2010)*, Valencia, Spain, pages 10-19, October 24-26, 2010.

[Harper and Blair, 2006] Harper, R. and Blair, A. Dynamically Defined Functions In Grammatical Evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (IEEE CEC 2006)*, pp. 2638-2645, 2006

[Hugosson *et al.*, 2010] Hugosson, J., Hemberg, E., Brabazon, A. and O'Neill, M. Genotype Representations in Grammatical Evolution. *Applied Soft Computing*, 10(1), 36-43, 2010

[Jefferson *et al.*, 1992] Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C. and Wang, A. Evolution as a Theme in Artificial Life: The Genesys/Tracker System. In Langton, C. G., Taylor, C., Doyne Farmer, J. and Rasmussen, S. (Eds.), *Artificial Life II* (pp. 549-578). USA: Addison-Wesley, 1992.

[Koza, 1992] Koza, J. R. *Genetic Programming: On the Programming of Computers by the Means of Natural Selectio*n. Cambridge, MA: MIT Press, 1992.

[Koza, 1994] Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press, 1994.

[O'Neill and Ryan, 2001] O'Neill, M. and Ryan, C. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4), 349–358, 2001.

[O'Neill and Ryan, 2003] O'Neill, M. and Ryan, C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. USA: Kluwer, 2003.

[O'Neill *et al.*, 2003] O'Neill, M., Ryan, C., Keijzer, M. and Cattolico, M. Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines* 4(1), 67-93, 2003.

[Robilliard *et al.*, 2006] Robilliard, D., Mahler, S., Verhaghe, D. and Fonlupt, C. Santa Fe Trail Hazards. *Lecture Notes in Computer Science*, 3871, 1-12, 2006

[Rothlauf and Oetzel, 2006] Rothlauf, F. and Oetzel, M. On the Locality of Grammatical Evolution. In Proceedings of the 9th European Conference on Genetic Programming, *Lecture Notes in Computer Science* (vol. 3905), 320-330. Budapest, Hungary: Springer, 2006.

[Sondahl, 2005] Sondahl, F. *Genetic Programming Library for NetLogo project*. Northwestern University. Available from http://cs.northwestern.edu/~fjs750/netlogo/final, 2005.

[Teahan, 2010a] Teahan, W. J. *Artificial Intelligence – Agent Behaviour I*. Ventus Publishing ApS, 2010.

[Teahan, 2010b] Teahan, W. J. *Artificial Intelligence – Agents and Environments*. Ventus Publishing ApS, 2010.

[UCD, 2008] UCD Natural Computing Research & Application Group. *Grammatical Evolution in Java (GEVA) Official Web Site*. Ireland, Dublin. Available from http://ncra.ucd.ie/geva, 2008.

[Wilensky, 1999] Wilensky, U. *NetLogo*. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University. Available from http://ccl.northwestern.edu/netlogo, 1999.