# Solving number sequence problems by genetic programming

| | |
|---|---|
| **Dozent**: | Ute Schmid |
| **Kurs**: | Kognitive Modellierung WS 2011/2012 |
| **Team 4:** | Fco. Javier Cano<br>Fernando Macias<br>Martina Milovec |

## Index

# 1. Introduction to genetic algorithms and genetic programming

Melanie Mitchell says in her book *An Introduction to Genetic Algorithms* that there is no right definition of “genetic algorithm”. Elements, like population of chromosomes, selection according to fitness, crossover to produce new offspring, random mutation of new offspring, are those which define genetics algorithm. Most important for genetic algorithms is fitness function that assigns a score to each chromosome in the current population (Mitchell 1997: 9). According to this function of a chromosome depends how the chromosome will solve problem.

How genetic algorithms and genetic programming are closely related will be explain more specific down below.

## 1.2 Genetic algorithms and genetic programming

Genetic programming is a branch of genetic algorithms. The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs as solution. Genetic algorithms create a string of numbers that represents the solution. Each possible solution is called individual or program.

Both of them follow the same steps to generate useful solutions to optimization and search problems, by an heuristic search that mimics the process of natural evolution. This is the Genetic Algorithm Basic Scheme:

```
1.Randomly create an initial population of programs from the available primitives
2.Repeat:
    3.execute each program and ascertain its fitness,
    4.select one or two program(s) from the population with a probability based on
      fitness to participate in genetic operations,
    5.create new individual program(s) by applying genetic operations with specified
      probabilities.
6.until an acceptable solution is found or some other stopping condition is met,
7.return the best-so-far individual.
```

## 1.3 Representation

In this project, the structure chosen to represent these programs is the Binary Tree, in which the variables and constants in the program (x, y and 2.2, 11 and 7) are leaves (also called terminals in GP). The arithmetic operations (+, - * and /) are internal nodes (or functions in GP). The sets of allowed functions and terminals together form the primitive set of a GP system.

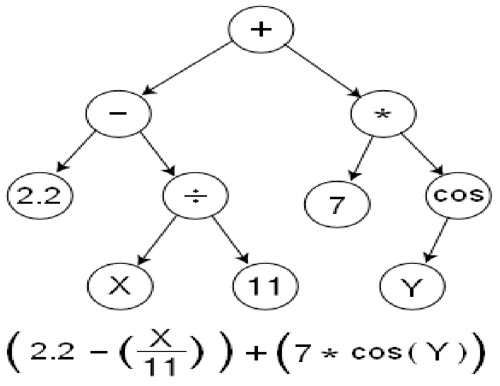

Figure 1: Binary tree

However, in order to represent these trees easily in paper, there is another widely used representation, which often makes it easier to see the relationship between (sub)expressions and their corresponding (sub)trees, known as Prefix Notation. For example, the expression '2.2 - x/11 + 7*cos(y)' becomes (+ (- 2.2 (/ x 11) (* 7 (cos y)))) when using Prefix Notation. Even further, since most of the times the arity (number of parameters) of each function in the function set is known, the brackets in prefix-notation expressions are redundant. Therefore, trees can efficiently be represented as simple linear sequences, e.g. the last expression could also be written unambiguously as the sequence < + - 2.2 / x 11 * 7 cos y >.

### 1.3.1 Steps in Tree Representation

The following part contains steps in Tree Representation which will be closely explain.

#### *1.3.1.1 Initialization*

Step in which the initial population of programs is generated, typically in a randomly fashion, but taking into account that the initial trees depth cannot exceed some specified maximum. There are three methods for initialization:

a) Full - the nodes are taken at random from the function set until the maximum tree depth is reached in the branches. Beyond that depth, only terminals can be chosen. This does not necessarily mean that all initial trees will have an identical number of nodes ('size' of the tree) or the same shape (this only happens when all the functions in the primitive set have an equal arity). This method always generates full trees, i.e. all leaves are at the same depth.

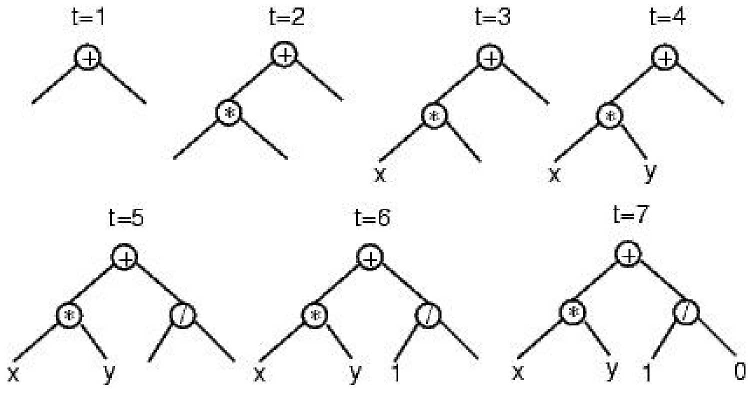

Figure 2: Full initialization

a) Grow - the nodes are selected from the whole primitive set (i.e., functions and terminals) until the depth limit is reached in any of the branches. After that, only terminals may be chosen. Because of this, this method allows for the creation of trees of more varied sizes and shapes.

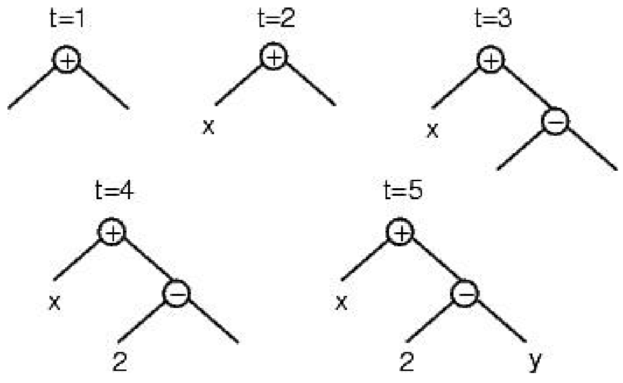

Figure 3: Grow initialization

b) Ramped Half-And-Half - this is the most widely used method, since neither the grow or full method provide a very wide array of sizes or shapes on their own. By using Ramped Half-and-Half, we generate 50% of the initial population using the Full Method and the remaining 50% is constructed using Grow Method. To help ensure that we generate trees having a variety of sizes and shapes, this method uses a range of depth limits (hence the term ramped).

### *1.3.1.2 Selection*

In this second step, the algorithm must choose some of the individuals from the whole population to be the 'parents' of the next generation. They are selected using the Fitness Function. This is some function that can produce a measure of the quality/accuracy of each program so they can be compared. E.g. the amount of time required to bring a system to a desired target state or the accuracy of the program in recognition of patterns/classifying objects. This is usually done by Tournament or by simple Fitness Evaluation, although any other method can be used too (Fitness-proportionate selection, stochastic universal sampling, etc.).

1. Tournament - easy to implement and provides automatic fitness rescaling, it is commonly used in GP. A number of individuals are chosen at random from the population. These are then compared with each other, and the best of them is chosen to be the parent (but it does not need to know how much better is one than the other). This method gives average-quality programs some chance of having children.

2. Fitness Evaluation - useful mechanism for giving a high-level statement of the problem's requirements to the GP system. It normally requires executing all the programs in the population, typically multiple times, so it is much more common to use an interpreter to evaluate the evolved programs. Interpreting a program tree means executing the nodes in the tree in an order that guarantees that nodes are not executed before the value of their arguments (if any) is known. This method could bias the search too much.

### *1.3.1.3 Offspring generation*

Using the parents selected from the population in the previous step, a new population is generated, and this will be the programs that the algorithm will use in the next iteration. This is where the biological inspiration appears, since we can use Crossover, Mutation and Reproduction between the parent programs:

a) Crossover - given two parents, randomly (and independently) a crossover point (node) is selected in each parent tree. Then, the offspring is created, by replacing the sub-tree rooted at the crossover point in a copy of the first parent with a copy of the sub-tree rooted at the crossover point in the second parent. Often crossover points are not selected with uniform probability. Typical GP primitive sets (using binary functions of arity 2) lead to trees with an average branching factor (the number of children of each node) of two, so the majority of the nodes will be leaves.

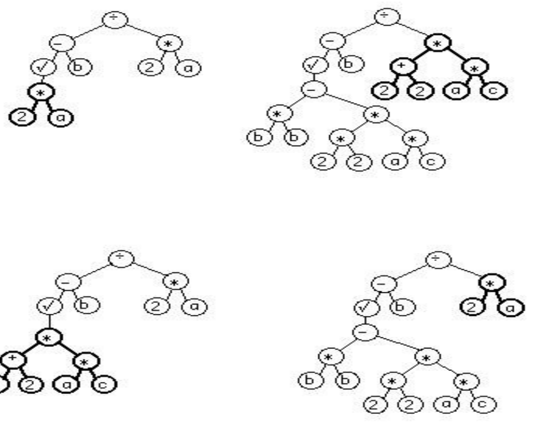

Figure 4: Crossover

b) Mutation - there are two methods to implement Mutation:

- Sub tree Mutation: given two parents, randomly selects a mutation point in a tree and substitutes that sub-tree rooted there with a randomly generated sub-tree. Sub-tree mutation is sometimes implemented as crossover between a program and a newly generated random program.

- Point mutation: a random node is selected and the primitive stored there is replaced with a different random primitive of the same arity taken from the primitive set. If no other primitives with that arity exist, nothing happens to that node. Point mutation is typically applied on a per-node basis (that is, each node is considered in turn and, with a certain probability, it is altered as explained above). This allows multiple nodes to be mutated independently in one application of point mutation.

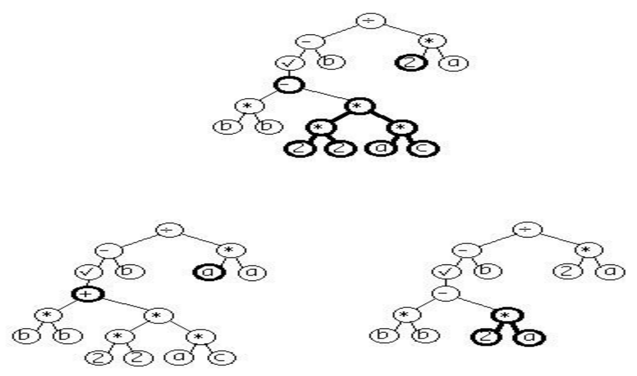
Figure 5: Mutation

c) Reproduction - this frequently used method appears when the rates (probability of being applied) of crossover and mutation add up to a value p which is less than 100%. In that case, an individual (program) is selected based on its fitness and then a copy of it is inserted into the next generation. Reproduction is used with a rate of 1-p, to reach the 100% of the Offspring Generation.

#### *1.3.1.4 Termination*

This type of algorithms always need a termination criterion. E.g: maximum number of generations to be run, or the error falls below one percent (highly used). Anyway, this criterion is strongly related to each specific problem.

#### *1.3.1.5 Solution designation*

For the solution designation, the best-so-far program is then harvested and designated as the result of the run, although one might wish to return additional data, e.g. the solution that this program produces to the given problem (in the case of the Number Series, the solution could be the function and the additional data the prediction of the next numbers in the series).

## 1.4 Conclusion

Genetic algorithms are very suitable to solve Number Series problems, specially using the tree representation, since any math function can easily be converted into a tree and vice versa.

# 2 Number sequences and empirical study

The following part contains the number sequences that are been used in empirical study and results of this empirical study.

## 2.1 Introduction

We always found it interesting how people are thinking and reasoning so this study about number predicting was very challenging for us. Finding out, if humans are better in solving number sequences problems then computer programs was mostly interesting. Humans are making decisions differently and in the most of the cases are impossible to predict the way of their thinking. Computer programs are different, humans are they who define how will a program work implementing they knowledge into a program which came out alone with conclusions.

## 2.2 Document number sequences

This part was for our study relevant, so we did small experiment with two groups of students which gave us interesting results.

For our empirical study we used the following number series:

a) 148, 84, 52, 36, 28

b) 2, 3, 5, 6, 7, 8, 10

c) 9, 20, 6, 17, 3

d) 2, 3, 5, 9, 17

e) 3, 7, 15, 31, 63.

Those number series were grouped according to their complexity into three levels: difficult ones, medium ones and easy ones. Therefore **a** and **b** are grouped into difficult ones, **c** into medium one, and **d** and **e** into easy ones.

For each number sequence we found following solution patterns which should be implementing into a system:

a) 148, 84, 52, 36, 28

Solution Pattern:

$x_1$=148 (example first number)

$x_n=x_{n-1} / 2 +10$ (example second number etc.)

b) 2, 3, 5, 6, 7, 8, 10

<u>Solution Pattern:</u>

$N= \{x|x=k2, \forall k\in N\}$

c) 9, 20, 6, 17, 3

<u>Solution Pattern:</u>

$x_1$=9 (example first number)

$x_n=x_{n-1} + (12)^{1-n\%2} – (15)^{n\%2}$

$x_n$= xn-1+11 if n%2=0xn-1-14 if n%2=1

d) 2, 3, 5, 9, 17

<u>Solution Pattern:</u>

$x_1=2$

$x_n=x_{n-1} \cdot 2 -1$

$x_n=x_{n-1} + 2^{n-1}$

e) 3, 7, 15, 31, 63

Solution Pattern:

$x_1=3$

$x_n=x_{n-1} \cdot 2 +1$

## 2.3 Document the empirical study

For our empirical study we had five test persons, aged between 20 and 24 years old. Our test persons are Spanish, and we grouped them into two groups. In first group were three females and in second group were two men who worked separately. Each group needed to solve above-named five number sequences. To bring some challenge into number series, we have decided to represent last number sequence (3,7,15,31,63) as words in German language (drei, sieben, fünfzehn, einunddreißig, dreiundsechzig). Our test persons are exchange students and they German skills are basic, so we wanted to see if they will be confused or if this presentation will cause them troubles solving number series. Two groups were working separately and were supervised. The duration of experiment depended of group. First group (females) needed more time then second group.

| | Number of persons | Genre | Age | Nationality | Profession | Duration |
|---|---|---|---|---|---|---|
| First group | 3 | female | 20-24 | Spanish | students | ca. 40 min |
| Second group | 2 | male | 23-24 | Spanish | students | ca. 30 min |

### 2.3.1 Procedure

Experiment was in Spanish language. Test persons were grouped at the beginning of experiment in two groups (first group was supervised by two supervisors, second was supervised by one supervisor) and they were working in the same room at different tables. Each of them had a paper with three number sequences on it (last two sequences were read number by number). Two groups were in a small introduction said different rules: first group must think aloud, second not, first group was allowed to discuss as a group, second group not (they should solve problems alone), the important was that each groups were making hypothesis and that they write these down. Time was not relevant for our study, so we have decided to give them so much time as they need. First group was also given some hints to help solve number problems. Following table shows how were the number sequences showed:

| | Number sequences: | Representation | Level |
|---|---|---|---|
| 1) | 148, 84, 52,36,28 | full | difficult |
| 2) | 9, 20, 6, 17, 3 | full | medium |
| 3) | drei, sieben, fünfzehn, einunddreißig, dreiundsechszig | full | easy |
| 4) | 2,3,5,6,7,8,10 | step by step | difficult |
| 5) | 2,3,5,9,17 | step by step | easy |

### 2.3.2 Results of the first group (3 females)

Duration of the experiment was about 40 minutes. First step was that they had a small introduction to solving number series, which included instruction how to express they thoughts loud, how to write down they solutions, how to consult with other group members, etc.

| | First group | | | |
|---|---|---|---|---|
| | Hypothesis | Next number prediction | Hint | Duration |
| 1.Number sequence | $x_n=(x_{n-1} / 2) +10$ | 24 | no | 4 min |
| 2.Number sequence | 1. subtracting 3 between first number and the one two positions forward<br>2. look at two first numbers and then subtract 3 to the number two positions backwards | 14 | no | 7 min |
| 3.Number sequence | 1. first number needs to be multiplicity by 2 and then add 1 | 127 | no | 3 min |
| 4.Number sequence | 1. x(n) = x(n-1) + 1<br>2. x(n) = x(n-2) + x(n-1)<br>3. a) x(n) = x(n-2) + x(n-1) + [Some constant: -1, -2, +1, ...]<br>b) an alternative: [x(n) = x(n-1) + 1] and [x(n) = x(n-2) + x(n-1)]<br>4. use the letters as in | 4<br>8<br>11 | yes | 15 min |

| | | | | |
|---|---|---|---|---|
| | the hint<br>5. use the letters as in the hint<br>6. a) use the letters as in the hint<br>b) write a pair amount of numbers, then skip one | 8<br><br>9 or 10<br><br>12<br><br>11 | | |
| 5.Number sequence | 1. x(n) = x(n-1) + 1<br>2. x(n) = x(n-1) + x(n-2)<br>3. a) x(n) = add all the numbers before – 1<br>b) x(n) = x(n-1) + x(n-2) + 1 (wrong, doesn't work for 2nd step!)<br>4. a) add all the numbers before - 1, Add all the numbers before - 2, Add all the numbers before - 3, and so on...<br>b) x(n) = x(n-1) * 2 - 1 | 4<br><br>8<br><br>18<br><br>14<br><br>33<br><br>33 | no | 10 min |

a) This number sequence was shown full and was categorized as difficult one, somehow finding a solution went very quickly. The group came to conclusion that division was necessary and that the missing operator was to add 10, so they predicted next number 24 correctly. All participants were making notes and their approach to challenges was motivated except one participant, who has for consequence that she was in the middle of work giving up to predict next number.

b) This number sequence was shown full and was categorized as medium one. Group needed more time to solve it as the first one. First they were thinking for themselves and it takes a few minutes to start proposing solutions. Subjects noticed the relation of subtracting 3 between first number and the one two positions forward. They hypothesis was that they need two first numbers and then they have to subtract 3 to the number two positions backwards. Finally group predicted the next number 14 correctly. The group spend more time trying to find a connection or more general hypothesis, finally they agreed about the above mentioned hypothesis.

c) This number sequence was shown full and was categorized as an easy one. First reaction of the group was surprising because was not numbers and it was on German language. Their first step was „translation“ into numbers and after that predicting next number 127 went easy.

Their hypothesis was that first number needs to be multiplicity by 2 and then add 1, which was correct.

d) This number sequence was shown number by number (step by step) and was categorized as difficult. After third hypothesis the group needed long time to give hypothesis and predictions at this point we decided to give them the hint with the series: O, T, T, F, F, S, S, E, N (one, two, three, four, five, six, seven) to make the group more 'open-minded', but this only made them get more confused, and they weren't sure. Besides, they tried to directly apply this concept of using the initials of the numbers which was the reason, why they had a hypothesis: Use the letters as in the hint. We noticed that the group was still confused with last hint, so we gave them another hint: „Try to think why some numbers are missing“. After that discussion was going into right direction. One subject came to solution: write a pair amount of numbers, then skip one -> which is a correct hypothesis (predicts that next skipped numbers will be 16 and 25) and completely different than the expected solution, what had for result that the group was more motivated.

e) This number sequence was shown number by number (step by step) and was categorized as an easy one. During this test, group came up with many hypothesis, some of them were completely wrong, but at the end, the group came up with two correct hypothesis, an expected one and a new one.

### 2.3.3 Results of the second group (2 males)

In this group two subjects were working separately and needed around 30 minutes to solve they number series. First they had a small introduction how to work with test, how they should be thinking and finding solutions for themselves. Unexpected they were solving number series very fast.

| | Second group | | | |
|---|---|---|---|---|
| | Hypothesis | Next number prediction | Hint | Duration |
| 1.Number sequence | 1. X(n)= X(n-1)/2<br>2. X(n) = X(n-1)/2+10 | 14<br>24 | no | 5 min |
| 2.Number sequence | X(n)= X(n-1)+11 if n %2== 0 else X(n)= X(n-1)-14 | 14, 0 | no | 3 min |
| 3.Number sequence | X(n)= 2^(n+1)-1 | 127 | no | 3 min |
| 4.Number sequence | 1. prime numbers<br>2. X_0=2;X_1=3; if ( n %2!=0) X(n)=X(n-1)+X(n-2) else X(n)=X(n-1)*X(n-2)<br>3. X_0=2;X_1=3; if ( n %2!=0) X(n)=X(n-1)+X(n-2) else X(n)=X(n-1)*X(n-2) | 18 | no | 15 min |
| 5.Number sequence | X_0=2; X(n)= X(n-1)*2-1 | 33 | no | 5 min |

a) Test persons look first for the differences between each consecutive number and then came to conclusion and predict next number correctly.

b) Prediction of the next number went quickly and easy.

c) Test person first translate numbers in Spanish language then into numbers, after that prediction went very easy.

d) Test persons after third hypothesis didn’t know how to predict next number, they get tired of finding a solution so both give up.

e) After the test persons had all five numbers, they came up with a hypothesis, before they didn’t try to find one.

# 3 ADATE

This part contains an Introduction to ADATE (Automatic Design of Algorithms Through Evolution), search procedure and solutions patters for some above mentioned number sequences.

## 3.1 Introduction

Adate is a system for automatic programming developed by Roland Olsson and automatic programming is inductive inference of algorithms. ADATE is classified into genetic programming as a branch of genetic algorithms. In this part we will try to explain how ADATE works and what kind of problems did appear during ADATE implementation into a system.

## 3.2 The ADATE Search

During the search ADATE builds a collection of programs called the kingdom, these programs are viewed as individuals. The kingdom expands adding new individuals. Individuals are generated by ADATE , which changes programs in the kingdom applying transformations to them. ADATE search individuals that are worse then already inserted programs into kingdom and replaces them with better one. The kingdom structure is a hierarchical, but based on the scientific classification, which is based on Linnean Taxonomy used by ADATE. Individuals that ADATE keeps in the kingdom are places into grids. ADATE use built complexity measures and places individuals by their syntactic and time complexity. The grids consists of a family, ADATE compares a newly generated individual with the individuals in the family and possibly replaces them (if this new individual has a better evaluation value then the old one). To evaluate value of individuals ADATE uses three different program evaluation functions which are internal. The user gives one output evaluation function that is used when calculating the three program evaluation functions, those functions calculate an evaluation value based on the number of correct and wrong training examples, the number of memory and time limit overflows and a user defined grade. To change individuals in the kingdom ADATE also uses transformations as the search operators, there are six transformations: R (Replacement), REQ (Replacement without making the individuals evaluation value worse), ABSTR (Abstraction), CASE-DIST (Case Distribution), EMB (Embedding) and Crossover. When one individual is selected for expansion, a sequence of transformations (above) is performed on it.

Next figure shows ADATE search process and describes the following steps.

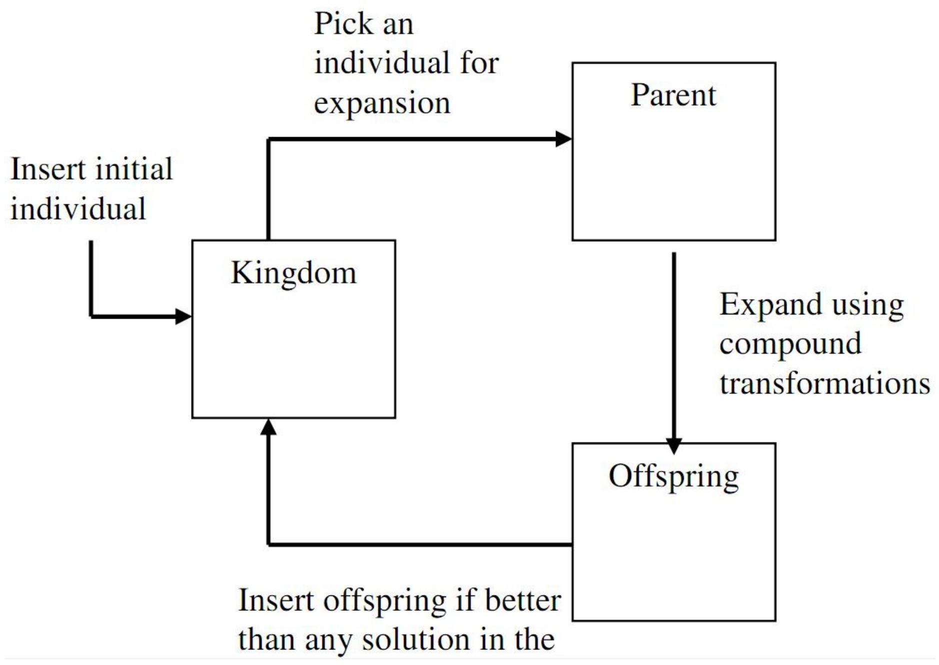
Pick an individual for expansion
Parent
Insert initial individual
Kingdom
Expand using compound transformations
Offspring
Insert offspring if better than any solution in the

Figure 6: ADATE search process

1. Insert the initial individual into the kingdom
2. Pick an individual from the kingdom for expansion and choose a cost limit
3. Generate a new individual with a compound transformation (sequence of transformations)
4. Insert the new individual in the kingdom if it is to form a new family or fits into a existing family
5. Repeat steps 3 and 4 a given number of times
6. Repeat steps 2 through 5 indefinitely.

### 3.3 Programming in ADATE-ML

ADATE-ML is a subset of Standard ML that is used by ADATE's inferred programs. This part will help us to solve algorithmic problems which include specification. Specification tells ADATE what a programm should do, but not necessarily how should do it. Specification consists of a text file containing ADATE-ML and ordinary ML code. ADATE-ML code contains a valid subset of ML in which programs are written. Specification has all informations needed to solve a problem, the code is embedded in ADATE's source code which is then compiled with the Mlton compiler. The specification consists of ADATE-ML code and ML code separated by the token '%%' because ADATE needs both code that can be used by the inferred function and code that it will use itself during the inference. In the follow step is shown an example of one specification.

```
ADATE-ML Code

%%

ML Code
```

## 3.5 Compiling ADATE

The specification code needs to be embedded in the ADATE source code before it is compiled. Makespec program process specification and the result from makespec should be an sml file that has the same name as the specification file with the suffix '.sml'. Next step is include this file in ADATE between main1.sml and main2.sml, then ADATE can be compiled with milton. After ADATE execute an important part is reading ADATE output files. After execute, three files are generate: log file, trace file and validation file. The log file contains all basic information about all individuals inserted into the kingdom. The trace file concerns itself with the kingdom and how it is organized in ADATE (statistics about the performing time of different tasks, time complexity grids and list of individuals that are queued for expansion. The validation file lists all individuals for the highest time limit for one grid. The best individual is usualy the best validated individual, which can be found at the bottom of validation file.

## 3.6 Solution patterns for number sequences

For the number series problem, the specification would be as easy as giving pairs input-output, and the definition in ML of the allowed operations.

The ADATE-ML part could be like this:

>> datatype input = input of int * int * int * int * int [depending on the input size, this definition could add more integers]

>> fun f ( ( X ) : input ) : int = raise D0

>> fun main ( ( X ) : input ) : int = f ( X )

>> %%

>> [ML code]

The ML code part should be, for each one of the number series:

1) 148,84,52,36,28

>> val Inputs = [ 148, 84, 52, 36, 28 ]

>> val Outputs = [ 24 ]

2) 2,3,5,6,7,8,10

>> val Inputs = [ 2, 3, 5, 6, 7, 8, 10 ]

>> val Outputs = [ 11 ]

Note: For this one, we need a bigger input, so the definition in the first line of the ADATE-ML code would change to:

>> datatype input = input of int * int * int * int * int * int * int

3) 9,20,6,17,3

>> val Inputs = [ 9, 20, 6, 17, 3 ]

>> val Outputs = [ 14 ]

4) 2,3,5,9,17

>> val Inputs = [ 2, 3, 5, 9, 17 ]

>> val Outputs = [ 33 ]

5) 3,7,15,31,63

>> val Inputs = [ 3, 7, 15, 31, 63 ]

```
>> val Outputs = [ 127 ]
```

Note: Obviously we can not input German words into ADATE, but this is similar to the human behavior (first translating into numbers, then operating)

## 4 Resources

Mitchell, M. (1997) *An Introduction to Genetic Algorithms.* Cambridge Publisher, Massachusetts

Web:

http://cswww.essex.ac.uk/staff/poli/gp-field-guide/toc.html
http://www.geneticprogramming.com/Tutorial/

http://www-ia.hiof.no/~rolando/ML/ADATE

http://mlton.org/