Finding Perceived Pattern Structures using Genetic Programming

Mehdi Dastani
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: mehdi@cs.vu.nl

Elena Marchiori
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: elena@cs.vu.nl

Robert Voorn
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: rbvoorn@cs.vu.nl

Abstract

Structural information theory (SIT) deals with the perceptual organization, often called the 'gestalt' structure, of visual patterns. Based on a set of empirically validated structural regularities, the perceived organization of a visual pattern is claimed to be the most regular (simplest) structure of the pattern. The problem of finding the perceptual organization of visual patterns has relevant applications in multi-media systems, robotics and automatic data visualization. This paper shows that genetic programming (GP) is a suitable approach for solving this problem.

1 Introduction

In principle, a visual pattern can be described in many different ways; however, in most cases it will be perceived as having a certain description. For example, the visual pattern illustrated in Figure 1-A may have, among others, two descriptions as they are illustrated in Figure 1-B and 1-C. Human perceivers prefer usually the description that is illustrated in Figure 1-B. An empirically supported theory of visual perception is the Structural Information Theory (SIT) [Leeuwenberg, 1971, Van der Helm and Leeuwenberg, 1991,
Van der Helm, 1994]. SIT proposes a set of empirically validated and perceptually relevant structural regularities and claims that the preferred description of a visual pattern is based on the structure that covers most regularities in that pattern. Using the formalization of the notions of *perceptually relevant structure* and *simplicity* given by SIT, the problem of finding the simplest structure of a visual pattern (SPS problem) can be formulated mathematically as a constrained optimization problem.

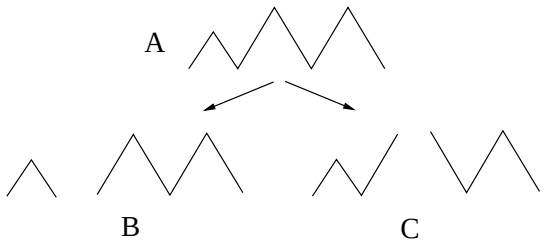

Figure 1: *Visual pattern A has two potential structures B and C.*

The SPS problem has relevant applications. For example, multimedia systems and image databases need to analyze, classify, and describe images in terms of constitutive objects that human users perceives in those images [Zhu, 1999]. Furthermore, autonomous robots need to analyze their visual inputs and construct hypotheses about possibly present objects in their environments [Kang and Ikeuchi, 1993]. Also, in the fields of information visualization the goal is to generate images that represent information such that human viewers extract that information by looking at the images [Bertin, 1981]. In all these applications, a model of gestalt perception is indispensable [Mackinlay, 1986, Marks and Reiter, 1990]. We focus on a simple domain of visual patterns and claim that an appropriate model of gestalt perception for this domain is an essential step towards a model of gestalt perception for more complex visual patterns that are used in the above mentioned real-world applications [Dastani, 1998].

Since the search space of possible structures grows exponentially with the complexity of the visual pattern, heuristic algorithms have to be used for solving the SPS problem efficiently. The only algorithm for SPS we are aware of is developed by [Van der Helm and Leeuwenberg, 1986]. This algo-

rithm ignores the important source of computational complexity of the problem and covers only a subclass of perceptually relevant structures. The central part of this partial algorithm consists of translating the search for a simplest structure into a shortest route problem. The algorithm is shown to have $O(N^4)$ computational complexity, where N denotes the length of the input pattern. To cover all perceptually relevant structures for not only the domain of visual line patterns, but also for more complex domains of visual patterns, it is argued in [Dastani, 1998] that the computational complexity grows exponentially with the length of the input patterns.

This paper shows that genetic programming [Koza, 1992] provides a natural paradigm for solving the SPS problem using SIT. A novel evolutionary algorithm is introduced whose main features are the use of SIT operators for generating the initial population of candidate structures, and the use of knowledge based genetic operators in the evolutionary process. The use of GP is motivated by the SIT formalization: structures can be easily described using the standard GP-tree representation. However, the GP search is constrained by the fact that structures have to characterize the same input pattern. In order to satisfy this constraint, knowledge based operators are used in the evolutionary process.

The paper is organized as follows. In the next section, we briefly discuss the problem of visual perception and explain how SIT predicts the perceived structure of visual line patterns. In Section 3, SIT is used to give a formalization of the SPS problem for visual line patterns. Section 4 describes how the formalization can be used in an automatic procedure for generating structures. Section 5 introduces the GP algorithm for SPS. Section 6 describes implementation aspects of the algorithm and reports some results of experiments. The paper concludes with a summary of the contributions and future research directions.

2 SIT: A Theory of Visual Perception

According to the structural information theory, the human perceptual system is sensitive to certain kinds of structural regularities within sensory patterns. They are called perceptually relevant structural regularities, which are specified by means of ISA operators: *Iteration, Symmetry* and *Alternations* [Van der Helm and Leeuwenberg, 1991]. Examples of string patterns that can be specified by these operators are *abab*, *abcba*, and *abgabpz*, respectively. A visual pattern can be described in different ways by applying different ISA operators. In order to disambiguate the set of descriptions and to decide on the perceived organization of the pattern, a simplicity measure, called *information load*, is introduced. The information load measures the amount of perceptually relevant regularities covered by pattern descriptions. It is claimed that the description of a visual pattern with the minimum information load reflects its perceived organization [Van der Helm, 1994].

In this paper, we focus on the domain of *linear line patterns* which are turtle-graphics, like line drawings for which the turtle starts somewhere and moves in such a way that the line segments are connected and do not cross each other. A linear line pattern is encoded as a letter string for which it can be shown that its simplest description represents the perceived organization of the encoded linear line pattern [Leeuwenberg, 1971]. The encoding process consists of two steps. In the first step, the successive line segments and their relative angles in the pattern are traced from the starting point of the pattern and identical letter symbols are assigned to identical line segments (equal length) as well as to identical angles (relative to the trace movement). In the second step, the letter symbols that are assigned to line segments and angles are concatenated in the order they have been visited during the trace of the first step. This results in a letter string that represents the pattern. An example of such an encoding is illustrated in Figure 2.

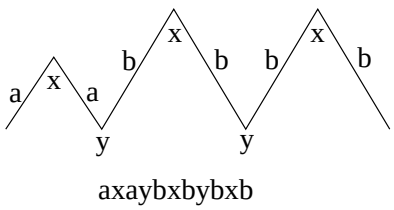

Figure 2: *Encoding of a line pattern into a string.*

Note that letter strings are themselves perceptual patterns that can be described in many different ways, one of which is usually the perceived description. The determination of the perceived description of string patterns is the essential focus of Hofstadter's Copycat project [Hofstadter, 1984].

3 The SPS Problem

In this section, we formally define the class of string descriptions that represent possible perceptually relevant organizations of linear line patterns. Also, a complexity function is defined that measures the information load of those descriptions. In this way, we can en-

code a linear line pattern into a string, generate the perceptually relevant descriptions of the string, and determine the perceived organization of the line pattern by choosing the string description which has the minimum information load.

The class of descriptions that represent possible perceptual organizations for Linear Line Patterns $\mathcal{LLP}$ is defined over the set $E = \{a, \ldots, z\}$ as follows.

1. For all $t \in E$, $t \in \mathcal{LLP}$
2. If $t \in \mathcal{LLP}$ and n is a natural number, then $iter(t, n) \in \mathcal{LLP}$
3. If $t \in \mathcal{LLP}$, then $symeven(t) \in \mathcal{LLP}$
4. If $t_1, t_2 \in \mathcal{LLP}$, then $symodd(t_1, t_2) \in \mathcal{LLP}$
5. If $t, t_1, \ldots, t_n \in \mathcal{LLP}$, then $altleft(t, < t_1, \ldots, t_n >) \in \mathcal{LLP}$ and $altright(t, < t_1, \ldots, t_n >) \in \mathcal{LLP}$
6. If $t_1, \ldots, t_n \in \mathcal{LLP}$, then $con(t_1, \ldots, t_n) \in \mathcal{LLP}$

The meaning of $\mathcal{LLP}$ expressions can be defined by the denotational semantics $[\![\ ]\!]$, which involves string concatenation ($\bullet$) and string reflection ($reflect(abcde) = edcba$) operators.

1. If $t \in E$, then $[\![t]\!] = t$
2. $[\![iter(t, n)]\!] = [\![t]\!] \bullet \ldots \bullet [\![t]\!]$ (n times)
3. $[\![symeven(t)]\!] = [\![t]\!] \bullet reflect([\![t]\!])$
4. $[\![symodd(t_1, t_2)]\!] = [\![t_1]\!] \bullet [\![t_2]\!] \bullet reflect([\![t_1]\!])$
5. $[\![altleft(t, < t_1, \ldots, t_n >)]\!] = [\![t]\!] \bullet [\![t_1]\!] \bullet \ldots \bullet [\![t]\!] \bullet [\![t_n]\!]$
6. $[\![altright(t, < t_1, \ldots, t_n >)]\!] = [\![t_1]\!] \bullet [\![t]\!] \bullet \ldots \bullet [\![t_n]\!] \bullet [\![t]\!]$
7. $[\![con(t_1, \ldots, t_n)]\!] = [\![t_1]\!] \bullet \ldots \bullet [\![t_n]\!]$

The *complexity function* C on $\mathcal{LLP}$ expressions, measures the complexity of an expression as the number of individual letters t occurring in it, i.e.

$C(t) = 1$
$C(f(T_1, \ldots, T_n)) = \sum_{i=1}^{n} C(T_i)$

During the last 20 years, Leeuwenberg and his co-workers have reported on a number of experiments that tested predictions based on the simplicity principle. These experiments were concerned with the disambiguation of ambiguous patterns. The predictions of the simplicity principle were, on the whole, confirmed by these experiments [Buffart et al., 1981, Van Leeuwen et al., 1988, Boselie and Wouterlood, 1989].

The following $\mathcal{LLP}$ expressions describe, among others, four different perceptual organizations of the pattern *axaybxbybxb*:

- $con(a, x, a, y, b, x, b, y, b, x, b)$,
- $con(symodd(a, x), y, symodd(b, x), y, symodd(b, x))$
- $con(symodd(a, x), iter(con(y, b, x, b), 2))$
- $con(symodd(a, x), iter(altright(b, < y, x >), 2))$

Note that these descriptions reflect four different perceptual organizations of the line pattern that is illustrated in Figure 2. The information load of these four descriptions are $11, 8, 6$, and 5, respectively. This implies that the last description reflects the perceived organization of the line pattern illustrated in Figure 2.

The *SPS problem* can now be defined as follows. Given a pattern p, find a $\mathcal{LLP}$ expression t such that

- $[\![t]\!] = p$ and
- $C(t) = \min\{C(s) \mid s \in \mathcal{LLP} \text{ and } [\![s]\!] = p\}$.

As mentioned in the introduction, the only (partial) algorithm for solving SPS problem is proposed by Van der Helm [Van der Helm and Leeuwenberg, 1986]. This algorithm finds only a subclass of perceptually relevant structures of string patterns by first constructing a directed acyclic graph for the given string pattern. If we place an index after each element in the string pattern, starting from the leftmost element, then each node in the graph would correspond to an index, and each link in the graph from node i to j corresponds to a gestalt for the subpattern starting at position i and ending at position j. Given this graph, the SPS problem is translated to a shortest route problem. Note that this algorithm is designed for one-dimensional string patterns and it is not clear how this algorithm can be applied to other domains of perceptual patterns. Instead, our formalization of the SPS problem can be easily applied to more complex visual patterns by extending the $\mathcal{LLP}$ with domain dependent operators such as Euclidean transformations for two-dimensional visual patterns [Dastani, 1998].

4 Generating $\mathcal{LLP}$ Expressions

In order to solve the SPS problem using genetic programming, a probabilistic procedure for generating $\mathcal{LLP}$ expressions, called BUILD-STRUCT, is used. This procedure takes as input a string, and generates a (tree structure of a) $\mathcal{LLP}$ expression for that string. The procedure is based on a set of probabilistic production rules.

The production rules are derived from the SIT definition of expressions, and are of the form $\alpha\ \ t_1 \ldots t_n\ \beta \longrightarrow \alpha\ \ P(t_1 \ldots t_n)\ \beta$

where α and β are (possibly empty) $\mathcal{LLP}$ expressions, $t_1, \ldots, t_n$ are $\mathcal{LLP}$ expressions, and P is an ISA operator (of arity n). The triple $(\alpha,\ \ t_1 \ldots t_n,\ \beta)$ is called *splitting* of the sequence.

A snapshot of the set of production rules used in BUILD-STRUCT is given below.

$\alpha\ t\ t\ \beta \longrightarrow \alpha\ iter(t,2)\ \beta$
$\alpha\ t\ iter(t,n)\ \beta \longrightarrow \alpha\ iter(t,n+1)\ \beta$
$\alpha\ iter(t,n)\ t\ \beta \longrightarrow \alpha\ iter(t,n+1)\ \beta$
$\alpha\ t_1\ t_2\ \beta \longrightarrow \alpha\ con(t_1,t_2)\ \beta$
$\alpha\ con(t_1,..,t_n)\ t\ \beta \longrightarrow \alpha\ con(t_1,..,t_n,t)\ \beta$
$\alpha\ t\ con(t_1,..,t_n)\ \beta \longrightarrow \alpha\ con(t,t_1,..,t_n)\ \beta$

A production rule transforms a sequence of $\mathcal{LLP}$ expressions into a shorter one. In this way, the repeated application of production rules terminates after a finite number of steps and produces one $\mathcal{LLP}$ expression. There are two forms of non-determinism in the algorithm:

1. the choice of which rule to apply when more than one production rule is applicable,
2. the choice of a splitting of the sequence when more splittings are possible.

In BUILD-STRUCT both choices are performed randomly. BUILD-STRUCT employs a specific data structure which results in a more efficient implementation of the above described non-determinism. The BUILD-STRUCT procedure is used in the initialization of the genetic algorithm and in the mutation operator.

We conclude this section with an example illustrating the application of the production rules system. The $\mathcal{LLP}$ expression $iter(con(a,b,a),2)$ can be obtained using the above production rules starting from the pattern *abaaba* as follows, where an underlined substring indicates that an ISA operator will be applied to that substring:

$\underline{aba}\ aba \longrightarrow con(a,b,a)aba$
$con(a,b,a)\ \underline{aba} \longrightarrow con(a,b,a)con(a,b,a)$
$\underline{con(a,b,a)con(a,b,a)} \longrightarrow iter(con(a,b,a),2)$

Note in this example that the *iter* operator is applied to two structurally identical $\mathcal{LLP}$ expressions (i.e. $\underline{con(a,b,a)con(a,b,a)} \longrightarrow iter(con(a,b,a),2)$). In general, the ISA operators are not applied on the basis of structural identity of $\mathcal{LLP}$ expressions, but on the basis of their semantics, i.e. on the basis of the patterns that are denoted by the $\mathcal{LLP}$ expressions (i.e. $\underline{symodd(a,b)con(a,b,a)} \longrightarrow iter(symodd(a,b),2)$).

5 A GP for the SPS Problem

This section introduces a novel evolutionary algorithm for the SPS problem, called GPSPS (Genetic Programming for the SPS problem), which applies GP to SIT. A population of $\mathcal{LLP}$ expressions is evolved, using knowledge based mutation and crossover operators to generate new expressions, and using the SIT complexity measure as fitness function. GPSPS is an instance of the generational scheme, cf. e.g. [Michalewicz, 1996], illustrated below, where $P(t)$ denotes the population at iteration t and $|P(t)|$ its size.

```
PROCEDURE GPSPS
t <- 0
initialize P(t)
evaluate P(t)
 WHILE (NOT termination condition) DO
  BEGIN
   t <- t+1
   WHILE (|P(t)|<|P(t-1)|) DO
     BEGIN
     select two elements from P(t-1)
     apply crossover
     apply mutation
     insert in P(t)
     END
  END
END
```

We have used the Roulettewheel mechanism to select the elements for the next generation. Therefore the chance that an element of the original pool is selected is proportional to its fitness. Since we apply our system to a minimization problem, the fitness function has to be transformed. This is done with the function $newF(element) = maxF(pool) - F(element)$. This ensures that the element with the lowest fitness will

have the highest probability of being selected. We have also made our GP elitist to guarantee that the best element found so far will be in the actual population.

The main features of GPSPS are described in the rest of this section.

5.1 Representation and Fitness

GPSPS acts on $\mathcal{LLP}$ expressions describing the same string. A $\mathcal{LLP}$ expression is represented by means of a tree in the style used in Genetic Programming, where leaves are primitive elements while internal nodes are ISA operators. The fitness function is the complexity measure C as it is introduced in Section 3.

Thus, the goal of GPSPS is to find a chromosome (representing a structure of the a given string) which minimizes C. Given a string, a specific procedure is used to ensure that the initial population contains only chromosomes describing the same pattern. Moreover, novel genetic operators are designed which preserve the semantics of chromosomes.

5.2 Initialization

Given a string, chromosomes of the intial population are generated using the procedure BUILD-STRUCT. In this way, the initial population contains randomly selected (representations of) $\mathcal{LLP}$ expressions of the pattern.

5.3 Mutation

When the mutation operator is applied to a chromosome T, an internal node n of T is randomly selected and the procedure BUILD-STRUCT is applied to the (string represented by the) subtree of T starting at n. Figure 3 illustrates an application of the mutation operator to an internal node. Observe that each node (except the terminals) has the same chance of being selected. In this way smaller subtrees have a larger chance of being modified.

It is interesting to investigate the effectiveness of the heuristic implemented in BUILD-STRUCT when incorporated into an iterated local search algorithm. Therefore we have implemented an algorithm that mutates one single element for a large number of iterations and returns the best element that has been found over all iterations. Although some regularities are discovered by this algorithm, its performance is rather scarce if compared with GPSPS, even when the number of iterations is set to be bigger than the size of the population times the number of generations used by GPSPS.

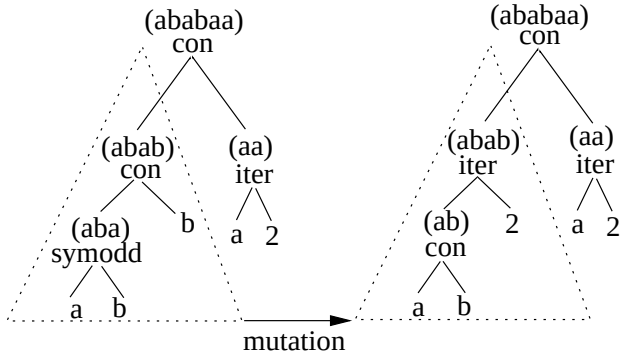

Figure 3: *Example of the mutation-operator.*

5.4 Crossover

The crossover operator cannot simply swap subtrees between two parents, like in standard GP, due to the semantic constraint on chromosomes (e.g. chromosomes have to denote the same string). Therefore, the crossover is designed in such a way that it swaps only subtrees that denote the same string. This is realized by associating with each internal node of the tree the string that is denoted by the subtree starting at that internal node. Then, two nodes of the parents with equal associated strings are randomly selected and the corresponding subtrees are swapped. An example of crossover is illustrated in Figure 4.

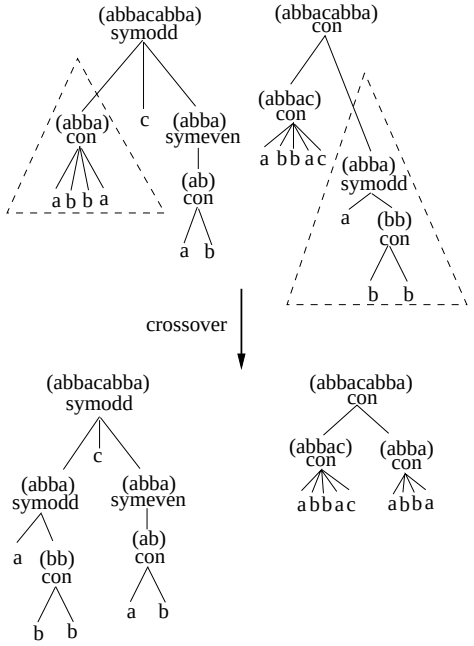

Figure 4: *Example of the crossover-operator.*

When a crossover-pair can not be found, no crossover takes place. Fortunately this happens only for a small portion of the crossovers. Usually there are more than one pair to choose from. This issue is further discussed in the next section.

5.5 Optimization

As discussed above, the mutation and crossover operators transform subtrees. When these operators are applied, the resulting subtrees may exhibit structures of a form suitable for optimization. For instance, suppose a subtree of the form $con(iter(b,2),a,con(b,b))$ is transformed by one of the operators in the subtree $con(iter(b,2),a,iter(b,2))$. This improves the complexity of the subtree. Unfortunately, based on this new subtree the expected $\mathcal{LLP}$ expression $symodd(iter(b,2),a)$ cannot be obtained.

The crossover operator is only helpful for this problem if there is already a subtree that encodes that specific substring with an *symodd* structure. This problem could in fact be solved by applying the mutation operator to the *con* structure. However, the probability that the application of the mutation operator will generate the *symodd* structure is small.

In order to solve this problem, a simple optimization procedure is called after each application of the mutation and crossover operators. This procedure uses simple heuristics to optimize the *con* structure. First, the procedure checks if the (entire) *con* structure is symmetrical and changes it into a *symodd* or *symeven* structure if possible. If this is not the case, the procedure checks if neighboring structures that are similar can be combined. For example, a structure of the form $con(c,iter(b,2),iter(b,3))$ can be optimized to $con(c,iter(b,5))$. This kind of optimization is also applied to $altleft$ and $altright$ structures.

6 Experiments

In this section we discuss some preliminary experiments. The example strings we consider are short and are designed to illustrate what type of structures are interesting for this domain. The choice of the values of the GP parameters used in the experiments is determined by the considered type of strings. Because the strings are short, a small pool size of 50 individuals is used. Making the size of the pool very large would make the GP perform better, but when the pool is initialized, it would probably already contain the most preferred structure. The number of iterations is also small to avoid generating all possible structures and is therefore set to 150. This allows us to draw preliminary conclusions about the performance of the GP.

Two important parameters of the GP are the mutation and crossover rates. We have done a few test runs to find a setting that produced good results. We have set the mutation-rate on 0.6 and the crossover-rate to 0.4. The mutation is deliberately set to a higher rate, because this operator is the most important for discovering structures. The crossover operator is used to swap substructures between good chromosomes.

We have chosen six different short strings that contain structures that are of interest to our search problem. Moreover, two longer strings are considered. For the two long strings the mutation and crossover rates above specified are used, but the poolsize and the number of generations are both set to 300. The eight strings are the code for the linear line patterns illustrated in Figure 5.

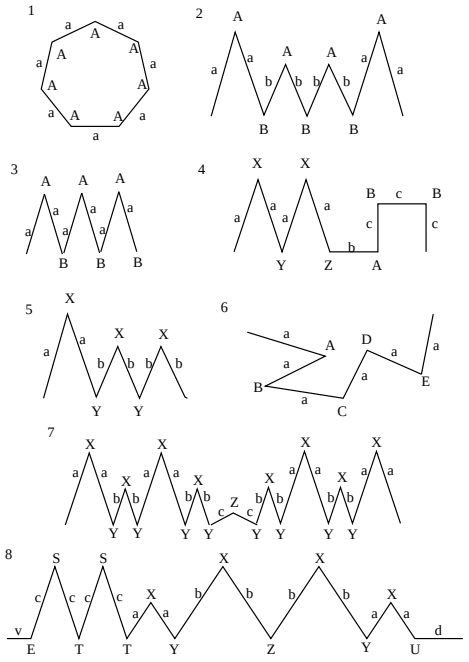

Figure 5: *Line drawings used in experiments.*

The algorithm is run on each string a number of times using different random seeds. The resulting structures are given in Figure 7, where the structure and fitnesses of the two best elements of the final population are reported. For each string GPSPS is able to find the optimal structure. The results of runs with different seeds are very similar, indicating the (expected) robustness of the algorithm on these strings.

Figure 6 illustrates how the best fitness and the mean fitness of the population vary in a typical run of GP-

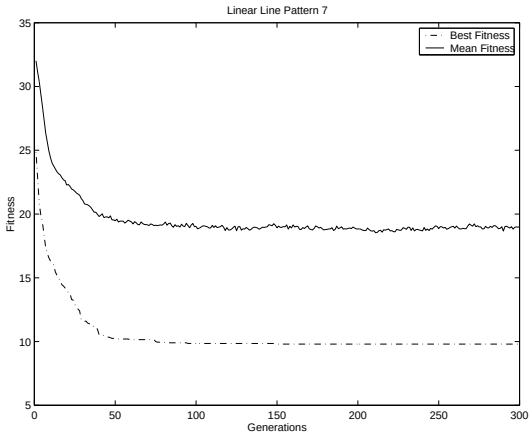

Figure 6: *Best and Mean Fitness.*

SPS on the line pattern number 7 of Figure 5. On this pattern, the algorithm is able to find a near optimum of rather good quality after about 50 generations, and it spends the other 250 generations to find the slighly improved structure. In this experiment about 12% of the crossovers failed. On average there were about 2.59 possible 'crossover-pairs' possible (with a standard deviation of 1.38) when the crossover operator was applicable.

The structures that are found are the most preferred structures as predicted by the SIT theory. The system is thus capable of finding the perceived organizations for these line drawings patterns.

7 Conclusion and Future Research

This paper discussed the problem of human visual perception and introduced a formalization of a theory of visual perception, called SIT. The claim of SIT is to predict the perceived organization of visual patterns on the basis of the simplicity principle. It is argued that a full computational model for SIT is computationally intractable and that heuristic methods are needed to compute the perceived organization of visual patterns.

We have applied genetic programming techniques to this formal theory of visual perception in order to compute the perceived organization of visual line patterns. Based on perceptually relevant operators from SIT, a pool of alternative organizations of an input pattern is generated. Motivated by SIT, mutation and crossover operations are defined that can be applied to these organizations to generate new organizations for the input pattern. Finally, a fitness function is defined that determines the appropriateness of generated organizations. This fitness function is directly derived from SIT and measures the simplicity of organizations.

In this paper, we have focused on a small domain of visual linear line patterns. The next step is to extend our system to compute the perceived organization of more complex visual patterns like two-dimensional visual patterns, which are defined in terms of a variety of visual attributes such as color, size, position, texture, shape.

Finally, we intend to investigate whether the class of structural regularities proposed by SIT is also relevant for finding meaningful organizations within patterns from biological experiments, like DNA sequences. For this task, we will need to modify GPSPS in order to allow a group of letters to be treated as a primitive element.

References

[Bertin, 1981] Bertin, J. (1981). *Graphics and Graphic Information-Processing*. Walter de Gruyter, Berlin NewYork.

[Boselie and Wouterlood, 1989] Boselie, F. and Wouterlood, D. (1989). The minimum principle and visual pattern completion. *Psychological Research*, 51:93–101.

[Buffart et al., 1981] Buffart, H., Leeuwenberg, E., and Restle, F. (1981). Coding theory of visual pattern completion. *Journal of Experimental Psychology: Human Perception and Performance*, 7:241–274.

[Dastani, 1998] Dastani, M. (1998). Ph.D. thesis, University of Amsterdam, The Netherlands.

[Hofstadter, 1984] Hofstadter, D. (1984). The copycat project: An experiment in nondeterministic and creative analogies. In *A.I. Memo 755, Artificial Intelligence Laboratory*, Cambridge, Mass. MIT.

[Kang and Ikeuchi, 1993] Kang, S. and Ikeuchi, K. (1993). Toward automatic robot instruction from perception: Recognizing a grasp from observation. In *IEEE Trans. on Robotics and Automation, vol. 9, no. 4*, pages 432–443.

[Koza, 1992] Koza, J. (1992). *Genetic Programming*. MIT Press.

[Leeuwenberg, 1971] Leeuwenberg, E. (1971). A perceptual coding language for visual and auditory patterns. *American Journal of Psychology*, 84:307–349.

[Mackinlay, 1986] Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. In *ACM Transactions on Graphics*, volume 5, pages 110–141.

[Marks and Reiter, 1990] Marks, J. and Reiter, E. (1990). Avoiding unwanted conversational implicatures in text and graphics. In *Proceeding AAAI*, Menlo Park, CA.

[Michalewicz, 1996] Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

[Van der Helm, 1994] Van der Helm, P. (1994). The dynamics of prägnanz. *Psychological Research*, 56:224–236.

[Van der Helm and Leeuwenberg, 1986] Van der Helm, P. and Leeuwenberg, E. (1986). Avoiding explosive search in automatic selection of simplest pattern codes. *Pattern Recognition*, 19:181–191.

[Van der Helm and Leeuwenberg, 1991] Van der Helm, P. and Leeuwenberg, E. (1991). Accessibility: A criterion for regularity and hierarchy in visual pattern code. *Journal of Mathematical Psychology*, 35:151–213.

[Van Leeuwen et al., 1988] Van Leeuwen, C., Buffart, H., and Van der Vegt, J. (1988). Sequence influence on the organization of meaningless serial stimuli: economy after all. *Journal of Experimental Psychology: Human Perception and Performance*, 14:481–502.

[Zhu, 1999] Zhu, S. (Nov, 1999). Embedding gestalt laws in markov random fields - a theory for shape modeling and perceptual organization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 21, No.11.

1	*string:* aAaAaAaAaAaAaA *structure:* a) iter(con(a,A),7) b) con(iter(con(a,A),2),iter(con(a,A),5)) *complexity* a) 2 b) 4
2	*string:* aAaBbAbBbAbBaAa *structure:* a) symodd(altleft(a,<A,con(B,symodd(b,A))>),B) b) symodd(con(symodd(a,A),altright(b,<B,A>)),B) *complexity* a) 6 b) 6
3	*string:* aAaBaAaBaAaB *structure:* a) iter(altleft(a,<A,B>),3) b) iter(con(symodd(a,A),B), 3) *complexity* a) 3 b) 3
4	*string:* aXaYaXaZbAcBcBc *structure:* a) altleft(symodd(a,X),<Y,altright(c,<con(Z,b,A),B,B>)) b) altleft(symodd(a,X),<Y, altright(c,<con(Z,b,A),symodd(B,c)>)) c) altleft(symodd(a,X),<Y,con(Z,b,A,c,iter(con(B,c),2))>) *scomplexity:* a) 9 b) 9 c) 9
5	*string:* aXaYbXbYbXb *structure:* a) altleft(a,<X,iter(con(Y,symodd(b,X)),2)>) b) altleft(a,<X,iter(altright(b,<Y,X>),2)>) *complexity:* a) 5 b) 5
6	*string:* aAaBaCaDaEa *structure:* a) altright(a,<altleft(a,<A,B>),C,D,E>) b) altleft(a,<A,B,C,D,con(E,a)>) *complexity:* a) 7 b) 7
7	*string:* axaybxbyaxaybxbyczcybxbyaxaybxbyaxa *structure:* a) symodd(con(iter(con(symodd(a,x), symodd(y,symodd(b,x))),2),c),z) b) symodd(con(iter(con(symodd(a,x), symodd(con(y,b),x)),2),c),z) *complexity:* a) 7 b) 7
8	*string:* vecsctcsctaxaybxbyzbxbyaxaud *structure:* a) con(v,altright(c,¡e,s¿),con(symodd(con(t,c),s), symodd(con(symodd(a,x),y,symodd(b,x)),z),u,d)) b) con(v,e,iter(altleft(c,¡s,t¿),2), symodd(con(symodd(a,x),y,symodd(b,x)),z),u,d) *complexity:* a) 13 b) 13

Figure 7: Results of experiments