Maintaining population diversity in Evolutionary Art

E. den Heijer[1,2] and A.E. Eiben[2]

[1]Objectivation B.V., Amsterdam, The Netherlands
[2]Vrije Universiteit Amsterdam, The Netherlands
eelco@objectivation.nl, gusz@cs.vu.nl ,
WWW home page: http://www.cs.vu.nl/~gusz/

Abstract. Evolutionary art is inherently more concerned with exploration than with exploitation, because users are typically more interested in evolving a collection of diverse images than converging to a single 'optimal' image. However, maintaining diversity is a difficult task. In this paper we investigate various techniques to promote population diversity in evolutionary art. We introduce customised mutation and crossover operators that perform a local search to diversify individuals and evaluate the effect of these operators on population diversity. We also investigate alternatives for the fitness crowding operator in NSGA-II; we use a genotype and a phenotype distance function to calculate the crowding distance and investigate their effect on population diversity.

1 Introduction

Evolutionary Art (EA) is a field that investigates ways to apply methods and ideas from Evolutionary Computation (EC) in the domain of generating aesthetically pleasing content. Determining the aesthetic value of an artefact in the EA system should be performed by one or more aesthetic measures or by one or more human beings, using Interactive Evolutionary Computation (IEC). Besides the ability to perform aesthetic evaluation, an EA system should also be creative. Margaret Boden defines creativity as the ability to create novel, surprising and valuable ideas [2]. In [3] Margaret Boden describes three 'roads to creativity'; combinational, exploratory and transformational. Combinational creativity is the process of coming up with novel ideas by combining existing ideas in unexpected ways. Exploratory creativity is the process of coming up with novel ideas by starting from an existing idea, and changing that idea in small steps to 'explore' the surrounding conceptual space for novel ideas. Transformational creativity is the process of altering the conceptual space, and is considered as the most radical, most difficult, and rare form of creativity. In our EA system, we try to establish a creative potential by using combinational and exploratory creativity. In order to achieve this goal, our search space (or concept space as Boden calls it) should be diverse at all times. In previous work we performed many experiments with unsupervised evolutionary art, and we have used a number of aesthetic measures as fitness functions. We have performed experiments

with a single aesthetic measures as a fitness function, in order to determine the 'style' of the aesthetic measures [9, 8] and we have investigated the combination of multiple aesthetic measures using a Multi-Objective Evolutionary Algorithm (MOEA) [10]. One of the findings in our work with MOEA was the issue of premature convergence and the subsequent lack of population diversity. We used the well-known NSGA-II [6] as the MOEA, and found that in many runs of the evolutionary algorithm the resulting Pareto front was 'dominated' by one or a few individuals, each having multiple offspring individuals that were visually similar to each other. The lack of population diversity is not unique to evolutionary art, and the issue has been investigated thoroughly in the EC literature. In this paper we want to investigate the application of methods and techniques that will promote and maintain population diversity in EA systems. Typical EC systems contain a phase of exploration followed by a phase of exploitation [5, **?**]. EC systems should exploit the building blocks of fit individuals in order to build new individuals that will score well on the fitness functions. On the other hand, EC systems should also maintain population diversity in order to evolve new individuals that may score even better in later generations. A lack of population diversity will result in (premature) convergence, whereby the population of individuals will be dominated by one or a few individuals. In this paper we postulate the assumption that unsupervised evolutionary art systems will benefit more from exploration than from exploitation. The underlying reason is that we think (like [1]) that aesthetic measures are more like heuristics than like actual metrics of aesthetic evaluation. The main goal of this paper is to investigate how we can promote and maintain population diversity in evolutionary art systems. The focus of our investigation is on the use of distance functions (calculating the distance between individuals in the population); we created custom genetic operators that maintain and enhance population diversity using distance functions, and we replaced the NSGA-II fitness crowding operator with one of our distance functions. Our research questions are the following:

1. Can we improve population diversity by using a custom crossover operator and a custom mutation operator?
2. Can we improve population diversity in a MOEA setup by replacing the standard NSGA-II fitness crowding operator with a genotypic/ phenotypic distance function?

This paper is structured as follows; in Section 2 we shortly describe existing techniques to increase population diversity. In our paper we calculate population diversity by calculating the distance between individuals and we describe a number of difference distance functions in Section 3. Our custom genetic operators are described in Section 4. We describe our experiments and their results in Section 5 and end with our conclusions in Section 6.

2 Population diversity

Population diversity in Evolutionary Computation refers to the amount of mutual difference between the individuals in the population. If population diversity

is low, then the difference between the individuals is low, and will be likely that offspring in the next generation will be similar to the individuals in the current population. When population diversity is low, an EC system is likely to converge to a sub-optimal solution. Maintaining population diversity in Genetic Programming (GP) systems has been investigated thoroughly [4, 5, 17]. We will briefly discuss techniques from literature that maintain diversity. In his first book on genetic programming, Koza [14] describes the well-known half-and-half ramped initialisation. In this initialisation scheme, half of the population is initialised using the 'full' method, and the other half is initialised using the 'grow' method. In the 'full' method each node is recursively initialised with a function from the function set until the maximum depth for the tree has been reached. All the leaves are initialised with a random terminal from the terminal set. With the 'growth' method, each node is either initialised with a function from the function set or a terminal from the terminal set. When one increases the tree depth during the initialisation of the population, the trees become larger and one speaks of a 'ramped' initialisation. Although the half-and-half ramped initialisation usually creates a diverse population of trees, there is no guarantee that there are no structural or behavioural duplicates in the population. Koza [14] therefore suggests (as does Jackson in [12]) to perform additional checks to verify that there are no duplicates in the initial population. The removal of structural duplicates may not be enough to ensure population diversity. Two genetic programs with different genetic tree structures may exhibit the same phenotypic behaviour. This may be caused by the presence of introns in the expression trees. Jackson [12] suggests to measure behavioural or phenotypic similarity in the initial population. In the EC literature there is a distinction between genotype diversity and phenotype diversity; we will describe them below.

2.1 Genotypic diversity

Genotypic diversity refers to the amount of mutual differences among the individuals in a population. In order to calculate the genotype diversity of a population, we need to calculate the difference or distance between two individuals. If one uses binary strings one can use the Hamming distance as a distance metric. If the genotype representation is a vector of reals, then one can use the Euclidean distance. But if one uses a tree representation, as is very common in genetic programming, then the calculation of the genotype distance become more complex. A number of techniques have been described in literature that calculate the difference or distance between two trees. In our implementation we use the tree distance metric from Ekárt & Németh [11], and we will describe it briefly in Section 3.1.

2.2 Phenotypic diversity

In NSGA-II population diversity is promoted by a crowding distance operator. This operator gives a penalty to individuals that resemble other individuals, and similarity between individuals is calculated as the difference between the scores

on the objectives of the individuals. This method is very generic but is not very useful in a creative EA system. Two individuals can have almost identical objective evaluations, but their phenotype/ image may look very different. In this case, the minor difference in fitness will significantly lower the possibilities of the individual with the slightly lower fitness to survive and/ or to reproduce. If the goal of the EA system is to evolve (or optimise) a single image, then this method works fine, but if the goal should be to evolve a collection of aesthetically pleasing images, then selection pressure should be lower, and diversity should be rewarded. We have implemented two distance functions based on image features and we will describe them in the Section 3.2.

3 Distance functions

In our custom genetic operators that we will describe in Section 4 we will use a number of distance functions to determine the similarity between two individuals (genetic programs) in the population. The distance can be based on genotype or structure (the expression tree of the program) or on phenotype (the result image of the program).

3.1 Genotype or structural distance

The structural distance metric by Ekárt and Németh is an efficient and fast metric for expression trees. The metric calculates the distance between two expression trees by performing a node by node comparison of the nodes of the expressions. If no node is present in one of the two expressions, a 'null' node is used in the comparison. The metric uses several rules for the different types of nodes (literals, functions, null etc.), and we refer to [11] for details.

3.2 Phenotype or image distance

We use two image distance functions and we will briefly describe them here.

Stricker & Orengo The first distance function is the Stricker & Orengo distance function [19]. This distance function computes the distance between two images I_a and I_b by calculating the distance between the two image feature vectors v_a and v_a, where

$$d(I_a, I_b) = \frac{\sum_{i=0}^{i<N} w_i \cdot |v_{a_i} - v_{b_i}|}{\sum_{i=0}^{i<N} w_i} \tag{1}$$

For the image features we used the average, standard deviation and skewness of the hue, saturation, brightness and colourfullness of the colour pixels of the image. Each image feature is assigned a weight w and the weights are shown in Table 1.

Image feature	Weight
Hue (avg)	4
Hue (sd)	4
Hue (skewness)	4
Saturation (avg)	1
Saturation (sd)	1
Saturation (skewness)	1
Brightness (avg)	2
Brightness (sd)	2
Saturation (skewness)	2
Colourfulness (avg)	2
Colourfulness (sd)	2
Colourfulness (skewness)	2

Table 1. Image features and their weights used in our Stricker & Orengo image distance function

Brightness distance We implemented a simple distance function based on the difference in brightness values of the pixels of two images. This distance function is more generic than the Stricker & Orengo function, since it disregards colour and only calculates the average distance in brightness;

$$d(I_a, I_b) = \frac{\sum_{x=0}^{x \leq w} \sum_{y=0}^{y \leq h} |b(I_a(x,y)) - b(I_b(x,y))|}{w \cdot h} \tag{2}$$

where w, h refer to the width, height of the image, $b(I_a(x,y)) \in [0..1]$ is the brightness of the pixel of image I_a at (x,y).

4 Custom genetic operators

In this section we describe our custom genetic operators crossover and mutation. Both operators determine whether a newly created individual is 'new' enough by calculating the distance between that individual and the rest of the population. The operators iterate until they have found an individual that has a distance that is higher than a predefined distance threshold t. If no such individual is found, the individual with the highest distance is used. The algorithm to calculate the distance between two individuals ($getDistance$) is used by both genetic operators, and is described in Algorithm 1.

4.1 Crossover

Population diversity can be maintained by introducing new genetic material in the population by the crossover operator. In previous experiments in the field of evolutionary art, we used the well-known standard subtree crossover [14]. In [13] Jackson added a local search mechanism to the standard subtree crossover mechanism that aims at improving population diversity. The idea of our custom

Algorithm 1 Algorithm that determines the distance between two individual; df=distance function, dt=distance threshold. This algorithm is used by our custom crossover and mutation

```
function getDistance( program1, program2, df, dt )
if ( pop=null ) then
  return 0;
end if
if df = structuralDistanceFunction then
  expression1 ← program2.getExpression()
  expression2 ← program1.getExpression();
  return df(expression1, expression2);
else
  image1 ← render(program1);
  image2 ← render(program2);
  return df(image1, image2)
end if
```

crossover is to do a local search after each crossover operation, and keep the child that differs enough from its parents;

$$\frac{d(child, parent_1) + d(child, parent_2)}{2} \geq t \tag{3}$$

where t is a predefined distance threshold. We use distance functions based on genotype distance and image distance. The algorithm for our crossover operator is presented in Algorithm 2.

4.2 Mutation

We also created a custom mutation operator (similar to our crossover and similar to the mutation in [13]) that does a local search upon each mutation step and picks first mutated offspring that is different enough from its parent. If no such individual is found, the individual with the highest distance is used. We present the algorithm in Algorithm 3.

5 Experiments and Results

We performed two experiments to investigate the effect of adding local search to our genetic operators on population diversity, and one experiment whereby we investigated the effect of replacing the NSGA-II fitness crowding operator.

5.1 Experiment 1: custom crossover

We created a specialised crossover that was inspired by [13] and was described in Algorithm 2. Our crossover uses a distance function to determine the genotypic/ structural or phenotypic/ behavioural distance between a parent and its child.

Algorithm 2 Algorithm for our custom crossover; c=child, p1=parent1, p2=parent2, df=distance function, dt=distance threshold; function *getDistance* is defined in Algorithm 1

```
function crossover( p1, p2, df, dt )
MAX_ATTEMPTS ← 20
attempts ← 0
bestSoFar ← null;
largestDistanceSoFar ← 00;
while attempts ≤ MAX_ATTEMPTS do
  child ← standardSubTreeCrossover(p1, p2);
  distance1 ← getDistance(child, p1, df, dt);
  distance2 ← getDistance(child, p2, df, dt);
  distance ← (distance1 + distance2)/2;
  if bestSoFar = null then
    bestSoFar ← child;
    largestDistanceSoFar ← distance;
  end if
  if distance > dt then
    return child
  end if
  if distance > highestDistanceSoFar then
    bestSoFar ← child;
    largestDistanceSoFar ← distance;
  end if
  attempts ← attempts + 1;
end while
return bestSoFar;
```

We created one version with the Ekárt & Németh distance function (which calculates genotypic/ structural distance), one version with the Stricker & Orengo distance function and one version with our brightness distance (both calculate image distance, thus phenotypic/ behavioural distance). In this experiment we initialised a small population of 51 individuals, and calculated all two-parent crossover combinations. We ignored performing a crossover between an individual with itself, so we had $51 \cdot 50 = 2550$ crossover operations. First, we performed crossover with a standard subtree crossover operator [14], and calculated the average fitness of the produced children, and the average distance between children and their parents. Next, we performed the same experiment with our three custom crossover operators.

From these numbers we can conclude in general that adding a local search to the crossover operator will improve population diversity; the mean genotype distance and phenotype distance is higher for each custom crossover when compared to the standard subtree crossover operator. A remarkable finding is that the increase in genotype diversity is higher when doing the local search on phenotype (using the local search with Stricker & Orengo, and also with our Brightness distance function) than when using local search with Ekárt & Németh. When doing

Algorithm 3 Algorithm for our custom mutation; c=child, p=parent, df=distance function, dt=distance threshold; function *getDistance* is defined in Algorithm 1

```
function mutate( p, df, dt )
attempts ← 0
MAX_ATTEMPTS ← 20
bestSoFar ← null;
largestDistanceSoFar ← 0;
while attempts ≤ MAX_ATTEMPTS do
  child ← standardSubTreeMutation(p);
  distance ← getDistance(child, p, df, dt);
  if bestSoFar = null then
    bestSoFar ← child;
    largestDistanceSoFar ← distance;
  end if
  if distance > dt then
    return child;
  else
    if distance > largestDistanceSoFar) then
      bestSoFar ← child;
      largestDistanceSoFar ← distance;
    end if
    attempts ← attempts + 1;
  end if
end while
return bestSoFar;
```

the local search with Ekárt & Németh both genotype and phenotype diversity increase when compared to the standard subtree crossover, but not as much as the increase when using local search using a phenotype distance function.

5.2 Experiment 2: custom mutation

We created 3 varieties of our mutation operator; all mutation operators operate according to Algorithm 3, but the differ in the distance function. The three distance functions that we used were 1) Ekárt & Németh tree distance, 2) Stricker & Orengo image distance and 3) our brightness image distance function. We initialised a random population of size 100 (using half-and-half ramped initialisation), and applied our custom mutation operator on each individual in the population. We performed 50 iterations of this setup (resulting in 5000 evaluations). For each parent-child pair we calculated the genotype distance using Ekárt & Németh tree distance metric, and we calculated the image distance using the Stricker & Orengo image distance. We calculated the mean distance (and the standard deviation) and present the results in Table 3. From this experiment we can conclude that all mutation operators with added local search using a distance function increase the mean distance between individuals, and

Crossover	Mean fitness child	Genotype Child-Parent distance (Ekárt & Németh)	Phenotype Child-Parent distance (Stricker & Orengo)
Standard Subtree	0.007 (0.041)	9.296 (4.010)	0.141 (0.084)
LS with Ekárt & Németh	0.007 (0.042)	9.367 (4.064)	0.142 (0.085)
LS with Stricker/ Orengo	0.006 (0.037)	9.378 (4.040)	0.177 (0.100)
LS with Brightness distance	0.006 (0.039)	9.569 (4.292)	0.160 (0.085)

Table 2. Results for different crossovers; the numbers are the mean values (and the standard deviation in parentheses) over 2550 evaluations

Mutation	Mean fitness child	Genotype Child-Parent distance (Ekárt & Németh)	Phenotype Child-Parent distance (Stricker & Orengo)
Standard subtree mutation	0.008 (0.043)	2.520 (5.445)	0.114 (0.127)
With Ekárt & Németh	0.007 (0.040)	12.630 (5.616)	0.171 (0.130)
With Stricker & Orengo;	0.006 (0.040)	6.899 (7.435)	0.248 (0.102)
With Brightness distance;	0.003 (0.027)	6.509 (7.399)	0.196 (0.123)

Table 3. Results for different mutations; we show the mean distances (and the standard deviation in parentheses) over 5000 evaluations

will result in a more diverse population. The addition of a genotype distance function in the mutation leads to more genetically diverse individuals (which is not really a surprise) but the individuals are also more diverse in their phenotype. However, the resulting mean image distance from the mutation operator with added Ekárt & Németh distance function is significantly lower than the mean image distance from the two mutation operators with the added image distance functions (Stricker & Orengo and Brightness distance). On the other hand, the mean genotype distance from the individuals resulting from the two mutation operators with added image distance operators are higher that the individuals created with the standard mutation, but lower than the individuals created with mutation operator with added Ekárt & Németh distance function.

5.3 Experiment 3: an alternative NSGA-II crowding operator

Our motivation for this investigation was the lack of population diversity in our previous experiments with unsupervised evolutionary art using multi-objective optimisation with NSGA-II [6]. Using the distance functions from Section 3 we performed an experiment in which we replaced the standard NSGA-II crowding operator with one of our distance functions. The NSGA-II fitness crowding operator assigns a score to each individual in a Pareto front based on the frequency of the evaluation values of the individual. Individuals that have a 'popular' combination of evaluation values will get a lower rating on fitness crowding. We performed a series of experiments using unsupervised evolutionary art using

NSGA-II, using three aesthetic measures as a fitness functions; the Ralph & Ross bell curve [18], the Global Contrast Factor [15] and Benford Law [7] (we used all aesthetic measures in previous experiments [10]). We tried four different setups; one setup used the standard fitness crowding operator (the standard used in NSGA-II), and in the other three we replaced the standard crowding operator by on of our three distance functions (see Section 3). The basic evolutionary parameters are given in Table 4.

Symbolic parameters	
Representation	Expression trees
Initialization	Ramped half-and-half (depth between 2 and 5)
Survivor selection	Tournament, Elitist (best 1)
Parent Selection	Tournament
Mutation	Subtree mutation
Recombination	Subtree crossover
Numeric parameters	
Population size	200
Number of runs	10
Tournament size	3
Crossover rate	0.90
Mutation rate	0.10
Maximum tree depth	8

Table 4. Evolutionary parameters of our evolutionary art system used in our experiments

We did 10 runs with each setup, and calculated the mean mutual distance in the Pareto front after 20 generations for each run. We calculated the mean genotype distance (using the Ekárt & Németh distance) and the phenotype/ image distance (using Stricker & Orengo). We present the mean distances and the standard deviation in Table 5. Looking at the results, we see that the use of

Crowding operator	Genotype distance (Ekárt & Németh)	Phenotype distance (Stricker & Orengo)
Standard fitness crowding	12.654 (0.699)	0.185 (0.025)
Ekárt & Németh	15.358 (1.532)	0.166 (0.033)
Stricker & Orengo;	13.808 (0.740)	0.189 (0.036)

Table 5. Results for different crowding operators; we show the mean distances (and the standard deviation in parentheses) over 10 runs

a different crowding operator has an influence on the population diversity. When using the genotype/ structure distance metric from Ekárt & Németh (instead of the standard fitness crowding function) we see that the mean structural distance increases (from 12.654 to 15.358), but mean image distance decreases (0.185 vs.

0.166). We suspect that the use of Ekárt & Németh as a crowding operator favours the development of offspring with introns; offspring with introns may have a high genotype distance, but may have a low phenotype distance. When using the image distance function by Stricker & Orengo, we see a small increase in mean image distance (from 0.185 to 0.189), but also an increase in mean tree distance (from 12.654 to 13.808).

6 Conclusions and Discussion

Our first research question was whether we could improve population diversity in evolutionary art system by using a custom crossover and mutation. Our results show that it is very difficult to add population diversity to evolutionary art system using a custom crossover operator using local search. Although the crossover operator using the genotype image distance function creates more diverse offspring than the standard crossover, the increase in diversity is modest at best. Using a phenotype distance local search does increase both genotype and phenotype diversity. We also investigated whether we could improve population diversity using a custom mutation operator. Our results confirm this; the offspring created with the various mutation operators are more diverse than offspring created using the standard subtree mutation operator. Our second research question was whether we could increase population diversity in a MOEA evolutionary art system using an alternative to the standard fitness crowding operator. Our results show that the use of a phenotype distance function is beneficial for maintaining both genotype and phenotype diversity in the Pareto fronts. Using a genotype distance function is beneficial for genotype diversity but not for phenotype diversity. We think the use of both genotype distance functions and phenotype (image) distance functions can also be beneficial for other components of evolutionary art systems. When used in selection for reproduction these distance functions could improve the population diversity by selecting only different parents (parents that have a high mutual distance) for crossover. This may lead to an inefficient crossover (a crossover that produces offspring with low fitness), so it should be investigated whether such a selection scheme is beneficial for both population diversity and search efficiency.

References

1. Steven Bergen and Brian J. Ross. Evolutionary art using summed multi-objective ranks. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 227–244. Springer New York, 2011.
2. Margaret Boden. *The Creative Mind.* Abacus, 1990.
3. Margaret Boden. *Creativity and Art: Three Roads to Surprise.* Oxford University Press, 2010.
4. Edmund Burke, Steven Gustafson, Graham Kendall, and Natalio Krasnogor. Advanced population diversity measures in genetic programming. In *Parallel Problem*

Solving from Nature PPSN VII, volume 2439 of *Lecture Notes in Computer Science*, pages 341–350. Springer Berlin / Heidelberg, 2002.
5. Edmund K. Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.
6. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2002.
7. Esteve del Acebo and Mateu Sbert. Benford's law for natural and synthetic images. In Neumann et al. [16], pages 169–176.
8. E. den Heijer and A. E. Eiben. Using aesthetic measures to evolve art. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.
9. E. den Heijer and A.E. Eiben. Comparing aesthetic measures for evolutionary art. In *Applications of Evolutionary Computation, LNCS 6025, 2010*, pages 311–320, 2010.
10. E. den Heijer and A.E. Eiben. Evolving art using multiple aesthetic measures. In *EvoApplications, LNCS 6625, 2011*, pages 234–243, 2011.
11. Anikó Ekárt and S. Németh. A metric for genetic programs and fitness sharing. In *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 259–270. Springer Berlin / Heidelberg, 2000.
12. David Jackson. Phenotypic diversity in initial genetic programming populations. In Anna Esparcia-Alczar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Uyar, editors, *Genetic Programming*, volume 6021 of *Lecture Notes in Computer Science*, pages 98–109. Springer Berlin / Heidelberg, 2010.
13. David Jackson. Promoting phenotypic diversity in genetic programming. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 472–481. Springer Berlin / Heidelberg, 2011.
14. J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. The MIT Press, Cambridge, MA, 1992.
15. Kresimir Matkovic, László Neumann, Attila Neumann, Thomas Psik, and Werner Purgathofer. Global contrast factor-a new approach to image contrast. In Neumann et al. [16], pages 159–168.
16. László Neumann, Mateu Sbert, Bruce Gooch, and Werner Purgathofer, editors. *Computational Aesthetics 2005: Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005, Girona, Spain, May 18-20, 2005*. Eurographics Association, 2005.
17. Thi Hien Nguyen and Xuan Hoai Nguyen. A brief overview of population diversity measures in genetic programming. In The Long Pham, Hai Khoi Le, and Xuan Hoai Nguyen, editors, *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, pages 128–139, 2006.
18. Brian Ross, William Ralph, and Hai Zong. Evolutionary image synthesis using a model of aesthetics. In *IEEE Congress on Evolutionary Computation (CEC) 2006*, pages 1087–1094, 2006.
19. Markus Stricker and Markus Orengo. Similarity of color images. In *Storage and Retrieval of Image and Video Databases III, Vol. 2*, pages 381–392, 1995.