Experiments in Constraint-Based Automated Scene Generation

Simon Colton

Department of Computing, Imperial College, London, UK
sgc@doc.ic.ac.uk

Abstract. We investigate the question of automatic scene construction for visual arts applications. We have implemented a system which can automatically infer a user's intentions from a partially formed scene, express the inferences as constraints, and then use these to complete the scene. We provide some initial experimental results with the system in order to compare two approaches to constraint-based scene construction. This leads on to a discussion about handing over increasingly meta-level responsibility when building computationally creative systems.

Keywords: scene generation, constraint solving, visual arts

1 Introduction

We have for some time been building an automated painter called The Painting Fool (`www.thepaintingfool.com`), which we hope will one day be accepted as a creative artist in its own right. We initially concentrated on automating various non-photorealistic rendering processes including segmenting and abstracting images, and the simulation of the look and usage of various natural media such as pencils, paints and pastels. Recently, however, we have investigated more cognitive, higher level issues related to painting. Firstly, we set up an expert system which can choose a painting style appropriate to an emotional keyword describing an image of a person's face. For instance, given the keyword 'sadness', The Painting Fool would choose to use a palette of muted colours and simulate pastels and pencils, in an attempt to heighten the melancholy expressed in the final painting. The painting style chosen in response to the keyword 'happiness' is very different. Moreover, we combined The Painting Fool with a vision system able to detect the emotion of a sitter for a portrait [5].

We have also addressed the question of getting The Painting Fool to invent its own scenes for painting. We have so far concentrated on constructing scenes with multiple similar objects arranged in some intelligent fashion. Paintings of such scenes are very common, for instance fields of flowers were very popular subject matter to the Impressionists. In [3], we implemented an evolutionary approach to the construction of such scenes, and we further used the HR system [2] so that the combined system both invented a fitness function and evolved a scene which maximised the fitness, with some surprising results. While this approach was successful, it had the drawback that to describe a new type of scene required defining a weighted sum of correlations over the elements in the scenes. In addition, certain constraints on the scene could not easily be guaranteed.

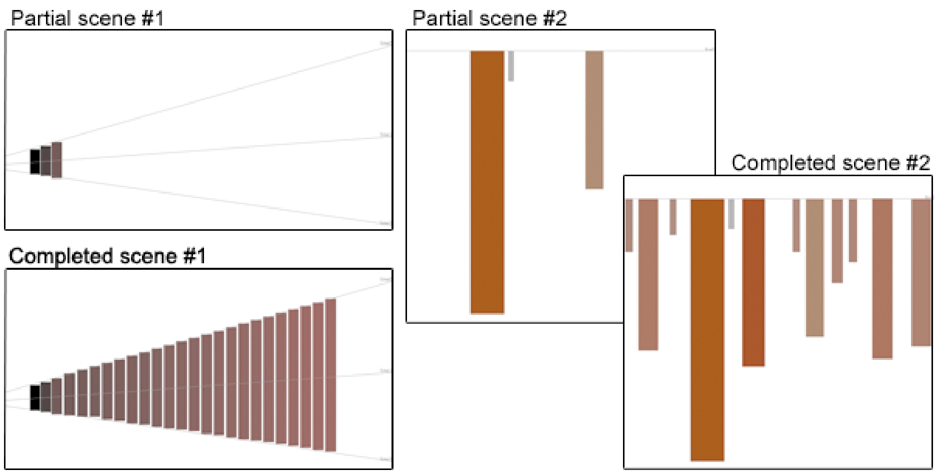

Fig. 1. Inspiring Scenes. In partial scene 1, three disjoint rectangles are arranged so that the tops, centres and bottoms are collinear, their widths and hues are equal, but both their brightness and saturation increase from left to right. In partial scene 2, three disjoint rectangles are arranged so that their tops are collinear and the rectangles with bases nearer the top of the scene are slimmer, shorter and less saturated than those with bases nearer the bottom of the scene.

We look here at an alternative approach to scene construction which attempts to address these drawbacks. In section 2 we describe how our system is able to induce a constraint satisfaction problem (CSP) from a partial scene supplied by a user. We also describe two ways in which the CSP can be used to generate scenes that contain elements adhering to the constraints exhibited in the partial scene. In section 3, we describe some initial experimental results from using the system. The work presented here has some overlap with the automatic derivation of designs from sketches [8]; diagrammatic reasoning [7]; geometric modelling via constraint solving [6] and scene understanding in machine vision applications [10]. However, our main inspiration has been Cohen's AARON program [9]. Much of the perceived creativity of AARON comes from the fact that it paints imagined scenes. In section 4, we describe some differences between the system we are building and AARON, and we use this as a springboard to discuss the requirement of handing over increasingly meta-level responsibility when building creative systems.

2 Constraint Based Scene Generation

We restrict ourselves to scenes which contain just rectangles of different locations, shapes, sizes and colours such as those in figure 1. These constructions are minimal, and can be considered placeholders for elements of a scene which will be painted. For instance, the rectangles in completed scene 2 of figure 1 could be replaced by images of tree trunks and painted to produce a woodland scene similar to that in Klimt's Beechwood Forest painting of 1903. It is certainly possible to evolve such scenes using the methods described in [3], so there is a degree of overlap with our previous evolutionary approach. However, unless the evolved scene completely maximised the fitness function, which is unlikely, there would be no guarantee that certain constraints required by the user would be upheld in the completed scene. For instance, having one rectangle fully or partially occlude another in scene 2 might ruin the aesthetic of the woodland scene. As described in section 5, we see the constraint approach and the evolutionary approach as complementing each other, and we envisage an overall system which is able to use these and other methods to generate sub-scenes in much larger, more complex scenes.

In addition to the guarantee of certain constraints holding, in order to rapidly train The Painting Fool with numerous scenes, another requirement we specified was that the user would train the scene generator visually. In practice, this means that the user moves and changes rectangles on screen until the constraints he/she wishes to be upheld are present in their partial scene. Then, the system infers various constraints and asks the user to discard any which are present but not desired (and allows the user to pre-empt this by turning off certain constraint deriving processes). In subsection 2.1, we describe how the constraints are derived and represented in the syntax of the CLPFD constraint solver [1], and in subsection 2.2, we describe how we introduce randomness to the scene description. The user dictates how many rectangles are needed to complete the scene, and to achieve this, the system uses the derived constraints to generate and solve either a single constraint satisfaction problem (CSP), or multiple CSPs, as described in subsection 2.3. In either case, the variables in the CSP relate to the x and y coordinates of each rectangle, along with their *height*, *width*, *hue*, *saturation* and *brightness*. The constraints relate the variables either of one rectangle (a *unary* constraint) or of two rectangles (a *binary* constraint). The ability to derive a specification for a scene further advances the work presented in [3], where the user had to explicitly specify a fitness function, or relied on the HR system to invent one. We plan to implement the ability to interpret the observed constraints as a fitness function, so that the evolutionary approach can also be driven by user supplied examples.

2.1 Deriving Constraints

We originally intended to use a descriptive machine learning system such as HR [2] to form a theory about the partial scene, and then mine the theory for potential constraints. However, we found that the kinds of constraints that need to be expressed are relatively simple, so theory formation was not needed. Instead, we wrote methods able to derive (a) constraints on the domains of variables (b) constraints expressing collinearity properties (c) constraints expressing general relationships between variables defining single rectangles and (d) constraints expressing relational properties between pairs of rectangles, as described below.

Domain restrictions

Any properties true of all the rectangles in a partial scene can be used to narrow down the domain of the variables. For instance, in partial scene 1 of figure 1, all the rectangles have the same width and hue. This can be expressed in a CLPFD constraint program by simply substituting the variable with the constant value. In addition, the system allows the user to constrain the values of variables to be within the bounds of those present in the partial scene. For instance, the heights of the rectangles in completed scene 2 of figure 1 are between the heights of the largest and smallest rectangles of partial scene 2, as the user specified this.

Collinearity constraints

The system looks at every triple of rectangles and checks for rough (i.e., within a radius of 20 pixels) collinearity of their centre points and eight points around their edges. For any line covering the same points on a triple of rectangles, it checks whether *all* the rectangles in the partial scene are similarly roughly collinear and if so, expresses this as a constraint. Any lines which are sufficiently similar to another are discarded, in order to keep the number of collinearity constraints down. In our early experiments, we found that having points fit exactly on lines often over-constrained the CSP. Hence, we relaxed the derived constraint to state that the y-coordinate is within 150 pixels of the line. Note that the scenes are of size 1000 by 1000 pixels, and – in line with standard graphics practice – the y-coordinate in scenes *increases* from top to bottom. As an example, in partial scene 1 of figure 1, the top left point of the three rectangles is $(66, 391), (94, 383)$ and $(122, 373)$ respectively from left to right. The system calculates the line between the first two points using the standard equation $(y - y_1) = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1)$. In order to keep the variables of the constraints whole numbers, the equation is multiplied throughout by $(x_2 - x_1)$. In our example, the line is therefore expressed as $28(y - 391) = (-8)(x - 66) \equiv 8x + 28y - 11476 = 0$. The system then checks whether the third point is within a distance of 20 pixels from that line. As this is true in our example, the system generates a constraint which dictates that the y-coordinate of the top left hand corner of each rectangle in the scene should be within 150 pixels of this line. The unary constraint in this case is expressed in CLPFD syntax as follows:

```
unary_constraint(line0,S) :-
    get_x(S,X), get_y(S,Y),
    (X*8) + (Y*28) - 11476 #> -150, (X*8) + (Y*28) - 11476 #< 150.
```

Note that `S` represents the rectangle to be constrained and `get_x` and `get_y` are able to to determine the x and y coordinate variables of the rectangle. There are similar predicates available for the height, width and colour variables.

Intra-rectangle relational constraints

At present, we have only one intra-rectangle relational constraint, which relates the height and width of the rectangles, so that the rectangles produced as solutions to the CSP have similar shapes. For example, in partial scene 2 of figure 1, the height to width ratio of the rectangles is between 1:5.16 and 1:7.71. To produce the completed scene, the user chose the option of constraining the proportions of all rectangles in the scene to be within these ratios. This constraint was expressed in CLPFD syntax as:

```
unary_constraint(height_width_ratio_min_max_516_771,S) :-
     get_height(S,H), get_width(S,W), H100 #= H * 100,
     (W * 516) #< H100, (W * 771) #> H100.
```

Inter-rectangle relational constraints

The system is also able to notice global relationships between the variables of pairs of rectangles. To do this, it first calculates various properties of each rectangle which supplement their defining properties. For each property P, it then calculates the sets: $R_{(P,=)} = \{(r_1, r_2) : P(r_1) = P(r_2)\}$, $R_{(P,<)} = \{(r_1, r_2) : P(r_1) < P(r_2)\}$ and $R_{(P,>)} = \{(r_1, r_2) : P(r_1) > P(r_2)\}$. Then, each instance of a subset relationship between these sets gives rise to a constraint. For instance, in partial scene 2 of figure 1, the following subset relationship was determined by the system: $R_{(bottommost-y,<)} \subseteq R_{(width,<)}$. That is, the system induced the hypothesis that if rectangle r_1 has a base that is higher in the scene than rectangle r_2, then r_1 also has less width than r_2. Remembering again that y-coordinates increase from top to bottom, this binary constraint is expressed as follows:

```
binary_constraint(bottom_above_implies_less_width,S1,S2) :-
    get_y(S1,Y1), get_y(S2,Y2), get_height(S1,H1), get_height(S2,H2),
    Bottom1 #= Y1 + H1, Bottom2 #= Y2 + H2,
    get_width(S1,W1), get_width(S2,W2),
    (Bottom1 #< Bottom2) #=> (W1 #< W2).
```

Note that constraints of this type are enabled by CLPFD's ability to post propositional constraints. The system found other such propositional constraints in partial scene 2, including the constraints that thinner rectangles are shorter and less saturated (and vice-versa). Note also that the system is able to detect global disjointness in a scene and derive an appropriate binary constraint for each pair of rectangles. It also determines the minimum amount of padding around each shape in the partial scene, and imposes this in the disjointness constraint, so that rectangles in the completed scene have similar padding.

2.2 Introducing Randomness

Unfortunately, the solutions to the kind of constraint problems which are generated tend to be fairly formulaic. For instance, the first scene presented in figure 2 is a valid solution to the constraints derived from partial scene 1 of figure 1. We see that the introduction of rectangles of the same shape, size and colour satisfies the constraints. In order to try to introduce more variety, such as that present in completed scene 2 of figure 1, we first experimented with an all-different constraint scheme and a constraint scheme which imposed a minimum difference between variables. However, we found that the former didn't introduce enough variety, and the latter tended to over-constrain the CSP. In addition, we also decided that for certain partial scenes, the user should be able to generate multiple

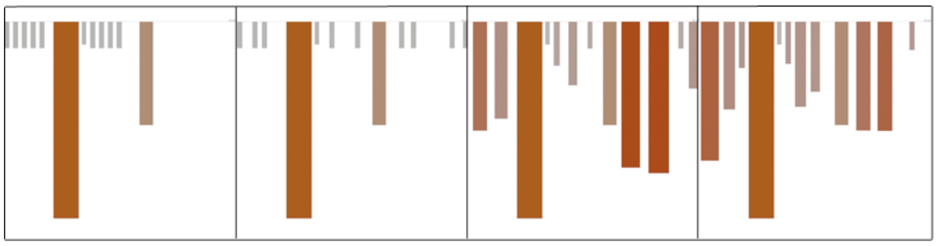

Fig. 2. Four completed scenes satisfying the constraints derived from partial scene 1 of figure 1. The first has no randomness imposed; the second has a random x-coordinate imposed and the final two have random x-coordinate, height and saturation imposed.

different scenes using the CSP. For these reasons, we allowed the user to specify that certain variables should have a random value ordering in the CSP search. As an example, to achieve the kinds of woodland scenes we desired from partial scene 2, we specified that the system should impose random value orderings on the height, saturation and x-coordinate of each rectangle. The completed scenes in figure 2 demonstrate the difference in visual variety achievable when the randomisation of variables is imposed. As described in section 3, we experimented to see whether reducing the domain of random values increased solving speed.

2.3 Using the CSP to Build Scenes

To recap, with our system, the user changes the shape, colour and location of a set of rectangles on screen, so that the partial scene exhibits the kinds of relationships they wish to see in the full scene. The system then induces certain constraints and the user can deselect any which are not there by design. Finally, the user chooses the number of additional rectangles, r, they wish to add, and the system is employed to complete the scene. We have experimented with two different ways to use the CLPFD solver to generate completed scenes. Firstly, we looked at trying to solve the scene generation problem in *one pass*. That is, a single CSP which contains variables for the shape, location and colour of r rectangles is written and solved. We impose a restart limit, which is useful when generating scenes with a random element. Secondly, we looked at a *sequential* approach where a constraint satisfaction problem is generated with variables for a single rectangle. This is solved and the rectangle is added to the scene, then the process iterates until the required number of rectangles are present. We impose a back-tracking parameter b: if a search for rectangle number n fails b times, then rectangle numbers $n-1$ and $n-2$ are removed from the scene, and the search starts again. When generating scenes with a random element, we found that over-constrainedness occurred when two rectangles with very similar values for the same variable were sequentially added to a scene. We further found that the scheme of removing both was more successful than removing just one of them, hence the reason for removing both rectangle $n-1$ and $n-2$.

3 Experimental Results

Our first set of experiments were designed to determine the fastest way to construct a completed scene from partial scene 1 of figure 1. We looked at both the one pass and the sequential approach, with the times taken to construct scenes with between 10 and 20 rectangles presented in figure 3. We see that, while the one-pass approach is faster for the addition of smaller numbers of rectangles, its solving time drastically increases when sixteen rectangles are required. Indeed, we found that constructing scenes with seventeen rectangles caused a memory problem which we have not yet overcome. In contrast, the sequential approach seems able to cope in a linear fashion with increasing numbers of rectangles, because it only has to solve a CSP for one rectangle at a time.

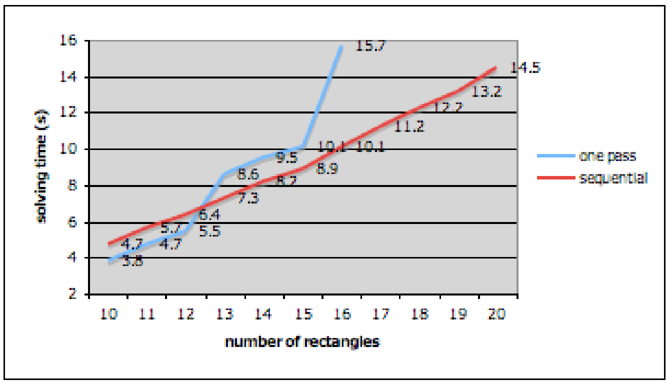

Fig. 3. Solving times for partial scene 1 for the one pass and sequential approaches.

Secondly, working with the constraints derived from partial scene 2 from figure 1, we experimented with various setups to determine the fastest way to construct a scene with twelve rectangles. In contrast to the first scene, scene 2 requires the x-coordinate, height and saturation variables to have a random value ordering, to achieve the desired visual variety. We experimented with different percentages for the size of the random variable domain, for instance, using a 17% random domain setting for a variable with integer domain d_1 to d_n means that a list of $0.17 * (d_n - d_1)$ values randomly chosen between d_1 and d_n without repetition becomes the value ordering for the variable. Using various settings for the backtracking search parameter and this domain percentage, we found that best time with the sequential approach was around 18 seconds. In contrast, we achieved much lower times for the one-pass approach. We experimented with various settings for the restart time (rs) and random domain percentage (rdp) and due to the random nature of the value orderings, for each pair $\langle rdp, rs\rangle$ which defined the search setup, we ran the system 10 times and recorded the average number of times the search was restarted and the average overall search time. The results are presented in table 1. We see that the lowest average time over a 10 run session was 2.6 seconds, which was achieved by both $\langle 20, 20\rangle$ and

$\langle 15, 75 \rangle$ settings. Also, the average time over all the setups is 7.6 seconds, which is less than half the time taken with the best sequential approach. We also note that – on average – the random domain percentage which is fastest over all the restart times is 75%, and on average over all the random domain percentages, using a restart time of 3 seconds was the fastest.

	Random domain percentage																					
Restart (s)	2		5		10		15		20		25		30		50		75		100		av.	
2	8.0	20.4	3.5	9.1	2.4	6.3	1.3	3.4	1.2	3.1	1.4	3.7	1.2	3.1	2.0	5.3	1.2	3.1	1.7	4.6	2.4	6.2
3	4.0	13.7	1.5	4.5	1.5	4.5	1.6	4.8	1.5	4.4	1.4	4.0	1.4	4.0	1.3	3.7	1.8	5.5	1.4	4.0	1.7	5.3
5	2.7	13.7	1.9	8.2	1.4	4.9	1.2	3.8	1.8	7.4	1.4	4.8	1.5	5.5	1.1	3.2	1.4	4.0	1.8	7.0	1.6	6.3
10	1.6	11.6	1.2	4.9	1.3	7.6	1.2	5.6	1.3	6.6	1.4	7.7	1.2	6.3	1.5	10.5	1.4	6.9	1.5	8.8	1.4	7.7
15	1.6	20.2	1.5	12.5	1.3	8.7	1.3	8.6	1.3	8.9	1.6	14.7	1.4	11.6	1.5	11.8	1.0	**2.6**	1.4	11.5	1.4	11.1
20	1.2	12.1	1.0	11.7	1.0	8.1	1.0	9.9	1.0	**2.6**	1.0	6.3	1.0	8.0	1.0	13.6	1.0	6.3	1.0	8.2	1.0	8.7
av.	3.2	15.3	1.8	8.5	1.5	6.7	1.3	6.0	1.4	5.5	1.4	6.9	1.3	6.4	1.4	8.0	1.3	4.7	1.5	7.4	1.6	7.6

Table 1. Number of restarts and solving times (in seconds) for the one pass approach. The best times (of 2.6s) are shown in bold face.

We intend to perform much more experimentation over a larger set of partial scenes, in order to be more concrete about the best settings for certain scenes. Our preliminary conclusions are that, for the construction of larger scenes, the sequential approach is advisable, but for smaller scenes of 15 rectangles or less, the one-pass approach may be quicker. We cannot yet be conclusive about the best settings for generating scenes with random aspects, because the two best settings from table 1 were for relatively large restart times, but on average over the random domain percentages, lower restart times seem to be more efficient. Also, as random domain percentages vary, there seems to be no obvious trend in efficiency, with the exception of 2%, which is clearly too small.

4 Discussion

Recall that our ambition for The Painting Fool is for it to ultimately be taken seriously as a creative artist in its own right. We believe that to achieve this will require addressing as many misconceptions in the art world (and society in general) as it will require the solving of technical problems. In particular, in [4], we argue that the default position – that machines cannot be creative – can lead to a vicious circle whereby the value of machine generated art is assessed not in terms of the artwork itself, but in terms of the (seemingly uncreative) process which produced it. As a start to breaking this vicious circle, we propose that the authors of creative software describe how it operates in terms of high level notions, rather than in terms of its algorithms. In particular, in [4], we suggest that any software which needs to be considered creative in order for its products to be assessed favourably should be described in terms of its skill, appreciation and imagination (which we call the *creative tripod*). Moreover, we suggest that the development of creative software could be guided by the creative tripod, i.e., we determine whether our software needs more behaviours which could be described as skillful, appreciative or imaginative. For instance, until we worked with emotion detection software (as described in [5]), we could not describe The

Painting Fool as appreciative, whereas now we can claim that the software has a greater appreciation of both its subject matter and the way in which its painting choices can affect the emotional content of its pictures. We are currently working on implementing behaviours which could be described as imaginative, and we are concentrating on enabling The Painting Fool to invent novel scenes in a similar way to the AARON program. The scene generation approach described here forms part of this programme of work.

If one were to find a criticism of AARON [9] as a creative system in its own right, it would be that Cohen has neither given it any aesthetic preferences of its own, nor the ability to invent new aesthetic preferences. In this sense, AARON remains largely an avatar of its author. While this is actually the wish of the program's author [personal communication], a general criticism of software which we may purport to be creative, is that it never exceeds its training. To combat this, we advocate the practice of *climbing the meta-mountain*, whereby once a creative system exhibits one behaviour, we examine the next creative responsibility that is still held by the human user of the system, and move on to automate behaviours which take over that responsibility. At each stage, we advocate implementing the ability to exceed our training with respect to the behaviour just implemented. For example, for a project with The Painting Fool, which resulted in the Amelie's Progress Gallery (see `www.thepaintingfool.com`), we implemented a number of different artistic styles (comprising a colour palette, abstraction level, choice of natural media, paint stroke style, etc.), and used this to paint numerous portraits of the actress Audrey Tatou. In addition, we gave The Painting Fool the ability to invent its own artistic styles, by choosing aspects of the style randomly. While this was simplistic, it was effective: around half of the 222 portraits in the gallery came from randomly invented styles and the expert system of artistic styles mentioned above is composed in part with styles which were randomly generated. As another case study, as previously mentioned in [3], we employed a machine learning approach to invent fitness functions, and the resulting system could be perceived as more imaginative, because we were unsure in advance not only of what the generated scene would look like, but also what aesthetic (fitness function) the scene would be evolved to maximise. Again, we can claim that the combined system exceeded our training somewhat.

As with the notion of the creative tripod, the notion of climbing a meta-mountain provides a useful way of communicating ideas about creative software to a general audience: in this case, we are describing how we train software to emulate artistic abilities and exceed their training. However, the notion of climbing a meta-mountain can also guide the development of creative software. To this end, we are working to a seven point meta-level implementation scheme, whereby we add behaviours which simulate (a) making marks on paper (b) making marks to represent scenes (c) painting scenes stylistically (d) choosing appropriate styles for scenes (e) inventing scenes (f) inventing scenes for a reason (g) evolving as an artist. We are currently working on projects which lie around (e) to (f) in this scheme, and we acknowledge that latter aspects of the scheme – in particular (g) – are not well defined yet.

5 Conclusions and Further Work

Via an initial implementation and preliminary experiments, we have shown that constraint-based scene generation is feasible, in particular that partial scenes can be defined visually and desired relationships can be guaranteed. We ultimately envisage a scene generation system which uses a combination of case-based, evolutionary, constraint-based and other approaches to allow the most flexibility. There are serious questions about the generality of the constraint-based approach, and we may find that it is only appropriate for certain types of scenes. In addition to the testing of the constraint based approach on many more scene types, we also plan to implement more constraint derivation methods to enable the building of more types of scenes. Following our methodology of training the system and then implementing behaviours which exceed our training, we will investigate methods to automatically change constraints derived from a partial scene, and to invent wholly new constraint schemes for the generation of scenes. As with the combined system presented in [3], we hope that with these additional abilities, the perception of imaginative behaviour in the system will be heightened.

Acknowledgements

We would like to thank the anonymous reviewers of this paper, who provided some very interesting feedback. We would also like to thank the reviewer of [3], who suggested that scene descriptions could be derived from exemplars, which led to the work presented here.

References

1. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proc. Prog. Languages: Implementations, Logics, and Programs*, 1997.
2. S Colton. *Automated Theory Formation in Pure Mathematics.* Springer, 2002.
3. S Colton. Automatic invention of fitness functions with application to scene generation. In *Proceedings of the EvoMusArt Workshop*, 2008.
4. S Colton. Creativity versus the perception of creativity in computational systems. In *Proceedings of the AAAI Spring Symp. on Creative Intelligent Systems*, 2008.
5. S Colton, M Valstar, and M Pantic. Emotionally aware automated portrait painting. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, 2008.
6. M Dohmen. A survey of constraint satisfaction techniques for geometric modeling. *Computers and Graphics*, 19(6):831–845, 1995.
7. M Jamnik. *Mathematical Reasoning with Diagrams: From Intuition to Automation.* CSLI Press, 2001.
8. P Jiantao and K Ramani. On visual similarity based 2D drawing retrieval. *Journal of Computer Aided Design*, 38(3):249–259, 2006.
9. P McCorduck. *AARON's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen.* W.H. Freeman and Company, 1991.
10. D Waltz. Understanding line drawings of scenes with shadows. In Patrick Winston, editor, *The Psychology of Computer Vision*, 1975.