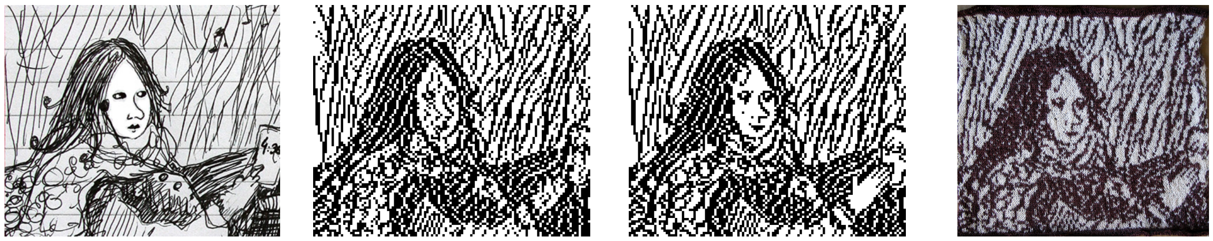
Generating Knitting Patterns from a Sketch: a CSP Approach

Marta Kryven* Elodie Fourquet†

University of Waterloo



(a) *Source drawing* **(b)** *Generated knitting pattern* **(c)** *User's edited version* **(d)** *Knitted version of (c)*

Figure 1: *A girl with an iPod use-case: a Fair-Isle knitted wearable image (d) generated using our system from a line-drawing image (a) converted to a knitted pattern (c).*

Abstract

Graphic patterns in knitting are composed of relatively large pixels and create a picture when seen from a distance, while on close viewing the image falls apart into its constituent stitches. Such patterns are constrained in use of colors due to the nature of the medium and in spacing between pixels as a durability concern and are a challenge to create. This paper shows how to convert an arbitrary line-drawing or photograph to a constraint-compliant Fair-Isle knitting pattern for a programmable knitting machine or a manual knitter by formulating it as a Constraint Satisfaction Problem (CSP). First we generate a constraint-inconsistent starting pixel assignment. Then we produce a perceptually similar constraint compliant solution, by minimizing and randomly distributing pixel flips to preserve gestalt features of the original design. We evaluate ways of generating a starting assignment using thresholding and dithering and of solving the problem using pseudo-random texturing and search: Random Walk, GSAT and Min-Conflict. Two hybrid solutions that achieve an improved design-dependent result are described. To test the algorithms an interactive knitting pattern generator was implemented.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—; I.3.6 [Computer Graphics]: Methodology and Techniques, Interaction Techniques—;

Keywords: non-photorealistic, constraint solving, knitting, sketch-based rendering, nontraditional media.

*e-mail:mkryven@uwaterloo.ca
†e-mail:ecdfourq@cs.uwaterloo.ca

1 Introduction

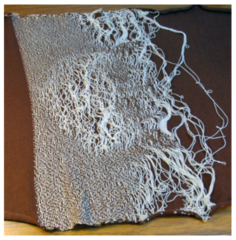
Visual patterns created by knitting depend on the physical structure of the medium, which defines and limits the possible images. A knitting pattern is an instructional diagram describing how to knit yarns to obtain an image on a finished garment. It shows sequences of stitches of each color that can be read by a human and reproduced either manually, or by a knitting machine. Having an instructional diagram is useful because knitting is a linear activity, while the knitted image that emerges is planar. There are several styles of multicolored knitting, in this paper we solve the problem of generating patterns for a style of knitting called Fair-Isle. It produces a non-reversible fabric, is supported by most knitting machines and easily made by hand (Figure 1).

Artists use digitally generated knitted patterns to visualize input images, algorithms and data. Figure 2 shows two examples. The easiest way to convert an arbitrary image into a knitting pattern is to lower pixel resolution and color depth, but that does nothing to control the comfort and durability of the result. Figure 2b shows the deleterious effect of designing an image at the front face of a knitted medium without taking into account the effects that occur the back of the fabric.

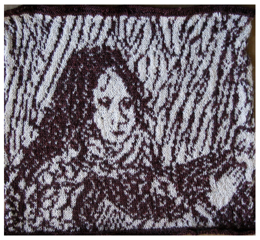
The front and back differ because a separate yarn is used for each color. While knitting with the active yarn, the other yarns, called floats, are hidden on the reverse. Long floats (Figure 2b) make a material catch and rip. Reasonable thickness and durability require that not more than two yarns are used in the same row and that same-color sequences are limited in length, usually between three and five stitches long [Radcliffe 2008]. Visually such patterns look dense. To achieve this easily most knitted designs limit themselves to geometric patterns with predictable spacing.

In this paper we describe and compare algorithmic solutions to converting a line drawing or a photograph into a durable knitting pattern. We discuss dithering, texturing and local search (Random Walk, Min-Conflict and GSAT), and describe two hybrid methods that achieve best results in terms of perceptual similarity between the original drawing and the actual knitted images. We used scanned drawings and photographs rather than vector graphics as sources because such images contain randomness, allowing us to generate knitted images that are more lively and dynamic [Arnheim 1974, p. 23–29] than conventional printed textiles.

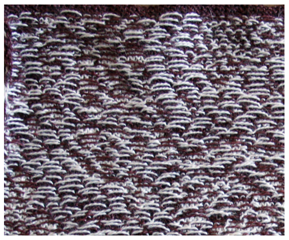
(a) *A knitted digitally processed photograph.*



(b) *The yarn on the back of (a).*



(c) *A knitted image made using our software.*



(d) *The yarn on the back of (c).*

Figure 2: *Front and back faces of a Fair-Isle knitted image. Images (a) and (b) are front and back of a ski mask designed by Andrew Salomone [Appendix A] using dithering. Images (c) and (d) are front and back of a pattern designed using our software.*

2 Related Work

While there are publications [Griswold 2007] and software, such as CorelDRAW, that address the problems of optimizing designs created by weaving, no similar attempts have been made for knitted patterns, which were traditionally made at home rather than industrially. Woven fabrics also adhere to constraints on float length and yarn color, but publications on weaving address generating and optimizing geometric patterns rather than arbitrary images and do not overlap with our subject. Some commercial knitting machine manufacturers, such as SilverReed, sell proprietary pattern editing software that can convert a user-specified clip-art into a format that can be read by the knitting machine. They do not offer pattern optimization.

An identical constraint on colors is used in grayscale half-tone printing. Half-tone printing is a way of rendering gradients of varied brightness by placing dots of dark ink close together on a light background, and is usually done by dithering. Dithering is a process of applying a brightness threshold matrix map that converts each pixel in an image to black or white depending on its position in the map and the brightness of its neighbours. Dithering places no constraints on the distance between pixels of the same color, but most dithering algorithms are designed to create either uniformly homogeneous [Bayer 1973] or uniformly unstructured [Floyd and Steinberg 1976] pixel patterns, in which the pixel structure is inconspicuous. In this paper we will take advantage of this property of dithering and dither gradient-containing images as a way of generating a starting assignment that partially satisfies knitting constraints.

A similar reduction of the color space is typical for pen-and-ink drawing[Salisbury et al. 1997], artistic hatching [Singh and Schaefer 2010], and images rendered with non-photorealistic line-drawing elements [Grabli et al. 2004]. When an image is converted to a black-and-white line drawing, some color information is removed and some semantic information is added because directional line shading and hatching traditionally represents movement [Cutting 2002]. Knitted images share the same effect: reducing the number of colors under the knitting constraints leads to semantic changes in the image. Moreover, as the space of structural elements constituting an image shrinks, the potential weight of symbolic meaning conveyed by the remaining elements increases [Arnheim 1974, p.47–95]. A common way to handle semantic artifacts is to choose different hatching algorithms for geometrically structured and unstructured sources [Ostromoukhov 2013]. Prominent edge structure in images benefits from using hatching algorithms that build on it, while stochastic algorithms are preferred in blurry images [Jodoin et al. 2002]. We take the same approach to knitting and differentiate between line-drawn and gradient-containing sources.

A problem of intelligent automation of an artist's work by inferring and solving a constraint satisfaction problem from an incomplete user-supplied scene has been addressed [Colton 2008; Colton and Ferrer 2012] for generating scenes with arrangements of multiple similar objects: wreaths, fields of flowers, woodlands. The distribution of good solutions in such scenes is dense and only a small number of initial conditions are provided. Additionally, it is assumed that there exist simple patterns of relationships between the color, hue, and size of the constituent objects in the partial user-supplied scene. Such patterns are extracted, formulated as constraints and iteratively reproduced with some randomization to generate an image.

On the other hand problems of 3D or 2D geometry modeling described using constraints with an empty initial assignment are usually not represented as a CSP. They are described using structures similar to L-Systems [Prusinkiewicz and Lindenmayer 1990] and graphic diagrams [Ijiri et al. 2005]. A knitted image is different because its final structure already exists in the initial assignment. Rather than generating an image from a recipe, we need only to improve its technical details to make it comply with the requirements of the physical medium.

Modeling of volumetric knitted garments has been done [Yuksel et al. 2012] for monochrome knitting. The authors describe a system for generating knitting instructions based on a 3D CAD user design. They then simulate the physical properties and appearance of the knitted garment. As a form of 3D printing knitting can be used to render surfaces with limited curvature as was demonstrated [Igarashi et al. 2008] in a system for generating hand-knitting instructions for rendering 3D models as knitted toys. Similar systems exist for generating instructions that model user-specified deigns with beads [Igarashi et al. 2012] and textiles [Mori and Igarashi 2007].

3 Generating A Starting Assignment

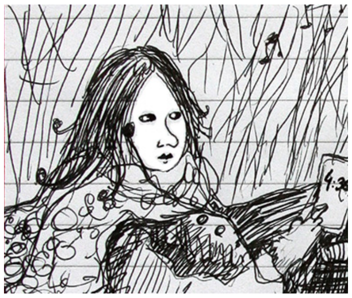
Formally a knitted image is a pixel grid of variables:

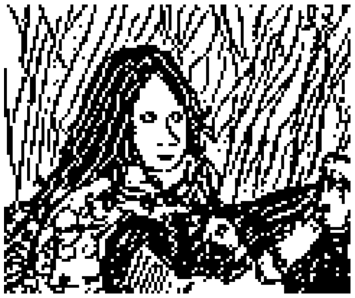
$$v_{i,j}\text{, where } i \in [1,n] \text{ and } j \in [1,m].$$

Fair-Isle knitting imposes two constraints on the pattern.

1. Only two colors can be used in one row, without loss of generality we will say that the two colors are black and white: $v_{i,j} \in \{\text{black, white}\}$
2. A float is less than f pixels long. Let:

$$\begin{aligned} l \in [1,n], k \in [1,n-1] \\ g(a_l) = true \iff a_1 = a_2 = ... = a_n \\ g(a_l) = false \iff \exists k, a_k <> a_{k+1} \end{aligned}$$


(a) *The source image*


(b) *Starting assignment using a brightness threshold. Variables: conflicts* 4532*, consistent* 62%.

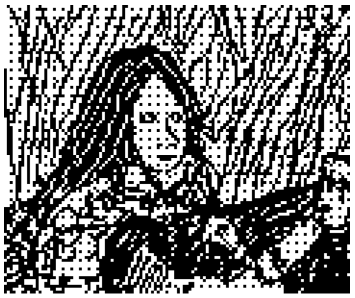
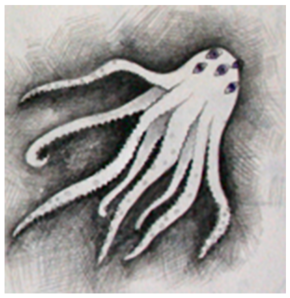
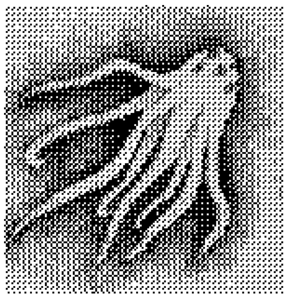

(c) *Starting assignment using ordered dithering. Variables: conflicts* 4210*, consistent* 64%.

Figure 3: *Starting assignments from a line-drawing. The size of the source image is* 120×100.


(a) *The source image*


(b) *Starting assignment using ordered dithering. Variables: conflicts* 5664*, consistent* 76%.

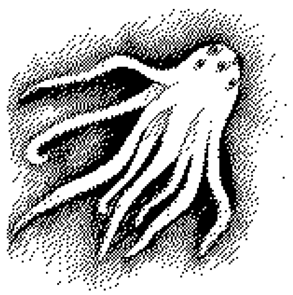

(c) *Starting assignment using Floyd-Steinberg dithering. Variables: conflicting* 1417*, consistent* 40%.

Figure 4: *Starting assignments from a gradient image. The size of the source image is* 151×155

then the constraint on float length is:

$$g(v_{i,j}, v_{i+1,j}, \cdots, v_{i+f,j}) = false \quad \forall i \in [1, n-f+1], j \in [1, m],$$

which, provided $v_{i,j}$ are boolean, can be implemented as:

$$v_{i,j} \oplus v_{i+1,j} \oplus \cdots \oplus v_{i+f,j} = false, \quad \forall i \in [1, n-f+1], j \in [1, m],$$

where the operator, $\oplus$, is an *exclusive or*.

The value of f depends on the thickness of the yarn and the desired strength of the material, in practice $f = 4$ is most common. Before we can proceed with solving the Constraint Satisfaction Problem (CSP) we need to generate S_0, a starting assignment to pixels $v_{i,j}$. We assume that the initial sketch is a monochrome image of the desired size, where each pixel in the resulting pattern corresponds to a knitted stitch.

If the input image is a line-drawing, as in Figure 3a, a starting assignment is best obtained using a pixel brightness threshold, Figure 3b is the result. If the input image is a continuous tone image, as in Figure 4a, it can be converted to a pixel image by dithering, using Floyd-Steinberg, Atkinson or ordered dithering (Figures 4b and 4c). The choice of dithering method affects the resulting texture. In most cases Floyd-Steinberg dithering is a reasonable neutral choice because it minimizes visual artifacts. Ordered dithering may be preferred for source images that contain a large empty space which the user prefers to fill in a structured way.

4 Solving the CSP

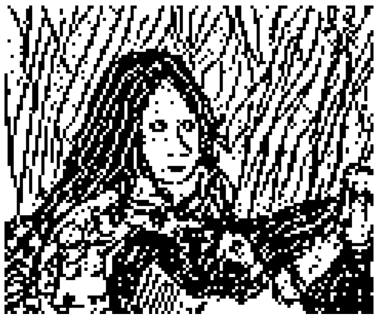
S_0 usually has floats longer than f and is thus inconsistent. To find a consistent assignment we search within the space of all complete assignments by flipping the value of one variable at a time starting at S_0 . Although the solutions are densely distributed, not all of them are equally good. A satisfactory solution should be as perceptually similar to the original image, thus an ideal algorithm would satisfy two objectives:

1. make changes that do not introduce spurious content;
2. minimize the amount of alteration.

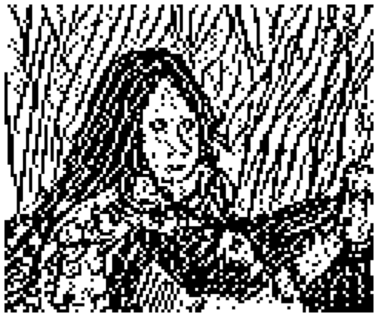
Next, we consider search algorithms that satisfy either (1) or (2), assuming that the constraints are hard and can not be partially satisfied. All local search implementation follow Algorithm 4.1, differing in the way conflicting variables are picked at each step.

Algorithm 4.1: LOCALSEARCHFORCSP($Assignment, numSteps$)

```
procedure LOCALSEARCHFORCSP($Assignment, numSteps$)
 if $Assignmentconsistent$
   then return ($Assignment$)
   else if $numSteps > maxSteps$
   then return ($notFound$)
   else { $x = pickConflictingVariable(Assignment)$
          $x.flipValue$
          $numSteps++$
          $LocalSearchForCSP(Assignment, numSteps)$
```



(a) $q = 20$ *Variables: conflicts* 2969, *consistent* 75%.



(b) $q = 5$ *Variables: conflicts* 1221, *consistent* 90%.

Figure 5: *Image 3b optimized with Random Walk.* $n = 100, m = 4, f = 4$. *Picking a smaller q results in an image that is noisier.*

Next, we describe and compare four ways of picking a conflicting variable.

4.1 A Variant of Random Walk

A Random Walk selects a variable at random at each step and flips its value if the variable is conflicting. This is repeated until a complete solution is found or a maximal number of iterations is reached. The implementation has two extra steps.

1. A variable is flipped only if, as a result, the number of violated constraints in decreased by a threshold m.
2. The algorithm terminates when fewer than q of the n randomly checked variables are found conflicting.

Step (2) implies that the algorithm stops at threshold level of constraint consistency, not attempting to generate a complete constraint-compliant assignment. However, the number of consistently assigned variables q is not a direct measure of the strength of the resulting material, because there is no guarantee that the longest floats were removed.

Comparing Figure 5a and Figure 5b shows that increasing q increases the amount of noise in the resulting image. The optimal amount of noise that a designer may want to introduce is image-dependent. It is possible to continue Random Walk iterations until all conflicts have been resolved and a complete solution has been found. Because variables are chosen randomly, successive runs produce different images and make different numbers of flips, which is also the case for Min-Conflict, discussed next.

4.2 Min-Conflict

Min-Conflict randomly chooses a conflicting variable, then selects a value for it that minimizes the number of violated constraints [Russell and Norvig 2009]. In the classic implementation, if no such value exists, Min-Conflict randomly picks any value that does not increase the number of violated constraints. Similarly, when choosing between several values that have the same impact it selects at random.

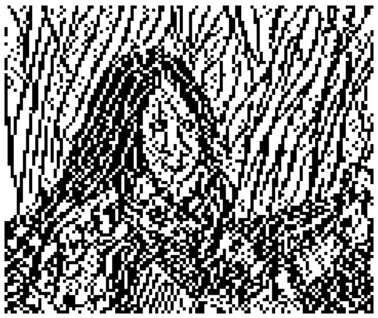


Figure 6: *Image 3b optimized with Min-Conflict,* $f = 4$. *Variables flipped:* 1119.

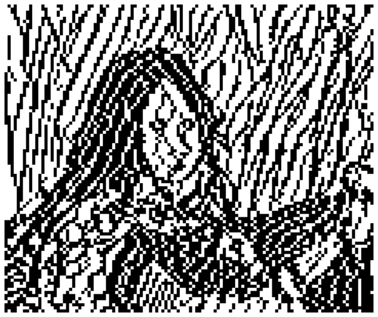
Our implementation is flip-aversive: it flips a variable only if the total number of violated constraints is reduced. Thus in our CSP Min-Conflict is analogous to a complete Random Walk, and its effect is identical to a Random Walk that runs until it solves the CSP. It is quite invasive, as shown in Figure 6. But, because Min-Conflict and Random Walk pick variables in a straightforward way, they have linear complexity.

4.3 GSAT

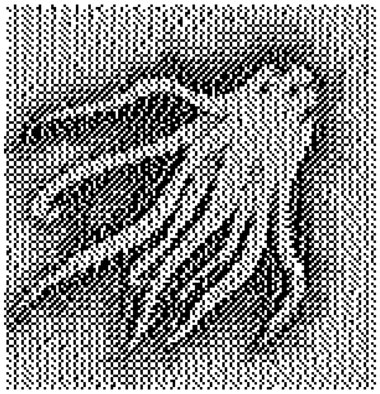
GSAT ranks variables by the number of constraints that are satisfied when a variable is flipped. Then it flips a top-ranked variable, which repairs the largest number of violated constraints [Hoos and Stützle 2000]. If several variables have equal rank, one is selected randomly. In our knitting CSP the order in which the top ranked variables are selected is unimportant.If a flip satisfying $2f + 1$ broken constraints is found, (Figure 8), it is selected immediately. Thus, the longest unsatisfied sequences that can be resolved with one flip are broken up first, then the second longest, and so on. Sequences that are too long to be satisfied with a flip are broken up $f + 1$ pixels at a time, near the end. As a result, GSAT flips the minimum number of variables.

A classic implementation of GSAT reranks all variables at each step and in the worst case has $O(n^2)$ complexity in the number of variables. In our CSP all variables involved in the same constraint clause also share the same row, therefore only the variables in the same row as the one just flipped are reranked. Thus, our implementation performs as $O(nw)$, where w is a length of a row.

Because most human-designed images contain prominent vertical and diagonal elements, it is common for pixels in a starting image to be vertically and diagonally aligned. Running GSAT in the presence of such features results in pixel flips that line up beside the pre-existing line structure which generate artifacts in uniformly colored areas (Figures 7a and 9a). GSAT artifacts are undesirable because people perceive visual images as collections of structural elements, the ease of understanding an image relies on a skeletal structure composed of (1) horizontal and vertical lines, (2) faces, (3) circles, (4) oblique lines, (5) simple curves, (6) simple geometric figures and (7) monochromatic shapes [Arnheim 1974, p.163–217]. A knitting pattern should avoid introducing these elements as artifacts and aim to preserve them in the original design. The difference between Figure 7a and Figure 7b shows that when GSAT is run starting from a structure of sufficient density the artifacts made

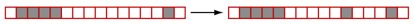


(a) $f = 4$. *Variables flipped:* 790.

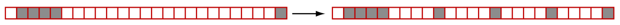


(b) $f = 4$. *Variables flipped:* 1031.

Figure 7: *Images 3b and 4c optimized with GSAT.*



(a) *The longest sequence that can be broken with one flip.*



(b) *GSAT breaks up long sequences every f pixels.*

Figure 8: *GSAT minimizations of variable flips, $f = 4$.*

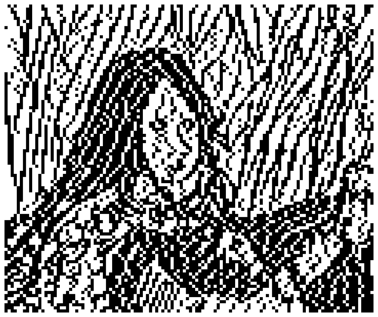
by GSAT blend in. In the next algorithm we exploit this property.

4.4 A Hybrid of Random Walk and GSAT

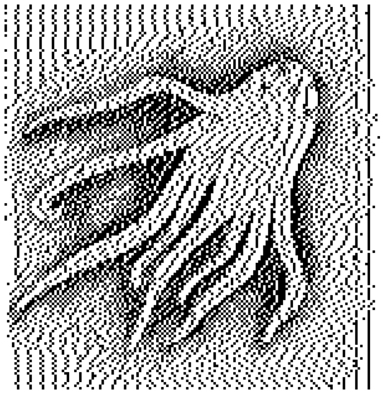
To address both objective (1) and objective (2) (Section 4), a method with much randomness and that does a minimal number of flips can be combined: first Random Walk for a number of steps, and then GSAT. By adjusting n, q, and m in Random Walk we control the amount of randomization. Note that setting $m = 2f + 1$ reduces the hybrid method to GSAT. Compared to Figure 10a optimized with GSAT, the hybrid method (Figure 9b) improves GSAT's handling of uniform areas and produces few line artifacts. The worst case performance of the hybrid method is equal to that of GSAT.

4.5 Pseudo-Random Texturing

Avoiding artifacts is hard(Figures 7a, 9, 10a). When long unsatisfied sequences are broken up at regular intervals, an uncontrolled regular pattern is added to the design. Instead of avoiding such artifacts we can sometimes control them and make them part of the resulting image. Algorithm 4.2 iterates through the unsatisfied sequences in order and breaks each of them starting at a random shift 1 to f pixels long to prevent aligning. It performs in linear time and makes less artifacts compared to GSAT as shown on Figure 10.



(a) *Image 3b, $q = 5$. Random Walk flips:* 204, *GSAT flips:* 572.



(b) *Image 4c, $q = 5$. Random Walk flips:* 4365, *GSAT flips:* 2262.

Figure 9: *Knitting patterns optimized with Random Walk+GSAT, $n = 100, m = 4, f = 4$.*

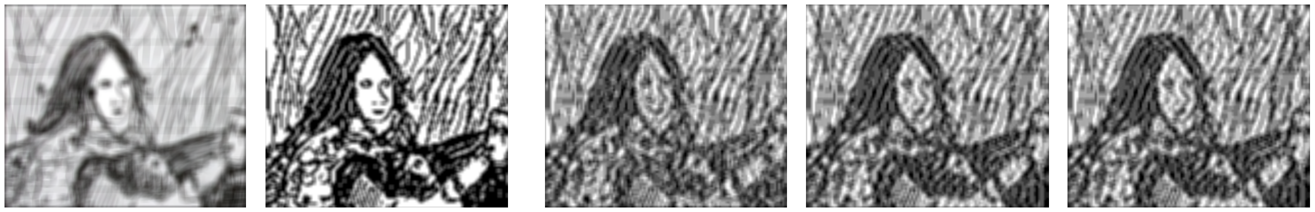
Algorithm 4.2: BREAKUPSEQUENCE($clause$)

```
procedure BREAKUPSEQUENCE($(clause)$)
  if $clauseconsistent$
    then return $(clause)$
    else $x = random(1, maxFloat - 1)$
    do $\begin{cases} x.flipValue \\ x = variableatx.position + maxFloat \end{cases}$
  while not $endOfSequence$
```

5 Implementation

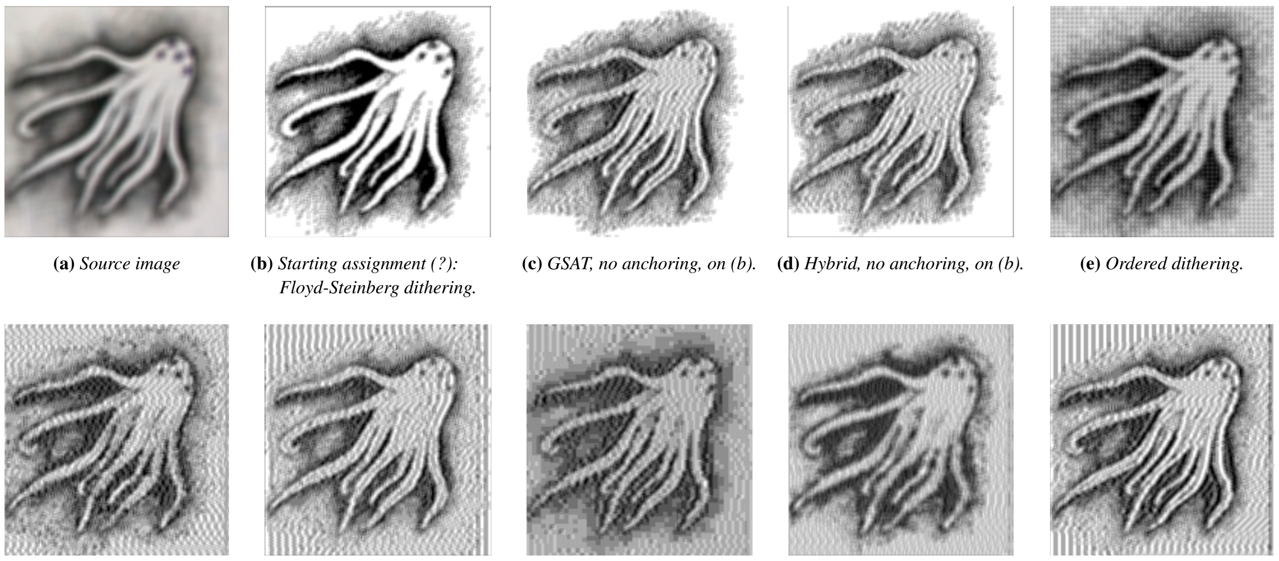
All algorithms described in this paper were implemented as open-source software for designing knitting patterns. It is available to download (Appendix A). The software has a fully automated mode suitable for new users, in which the user selects an image and accepts defaults that perform reasonably well on average. More experienced users can configure algorithm parameters manually. Pixel and dithering thresholds, dithering type, image orientation (horizontal or vertical), random walk duration, float length, and whether both yarns are anchored at the edges can be selected interactively. The image may be reoriented horizontally or vertically to pick a different knitting direction or to start with a more constraints satisfied. Yarn anchoring produces a fabric of a uniform thickness, and is required by some knitting machines. However, the user may want to turn off yarn anchoring for manual knitting, substantially reducing the number of flips.

After an initial assignment is generated, the user can interactively



(a) *Source image* **(b)** *Starting assignment: thresholding.* **(c)** *Min-Conflict on (b).* **(d)** *GSAT on (b).* **(e)** *Hybrid on (b).*

Figure 11: *Comparing the generated knitting patterns to the original drawing using Gaussian blur.*



(a) *Source image* **(b)** *Starting assignment (?): Floyd-Steinberg dithering.* **(c)** *GSAT, no anchoring, on (b).* **(d)** *Hybrid, no anchoring, on (b).* **(e)** *Ordered dithering.*

(f) *Pseudo-random on (b)* **(g)** *Hybrid on (b)* **(h)** *GSAT on (e).* **(i)** *Atkinson dithering with pseudo-random texturing.* **(j)** *GSAT on (b).*

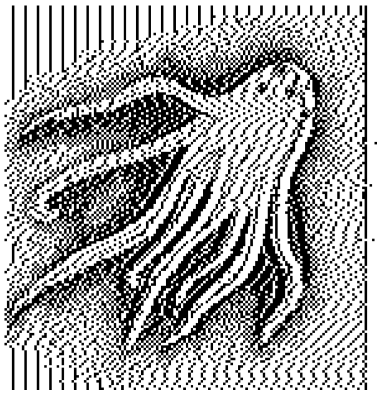
Figure 12: *Comparing the generated knitting patterns to the original drawing using Gaussian blur.*

edit it and flip individual pixels to add details, correct pixelation artifacts, or remove clutter. The user can dynamically zoom in for ease of editing and zoom out to see how the image from a distance. The image statistics—number of remaining conflicts and percentage of satisfied assignments—is displayed for guidance. The user then selects an optimization algorithm. It is useful to try several configurations because aesthetic qualities depend on the density, gradients and content of the starting image.

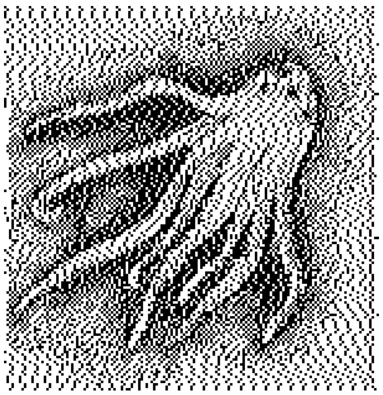
After the constraint solver has run, the user can again interactively edit the image. The user may choose to relax constraints in visually important areas, such as faces. An example of a knitting pattern with several constraints deliberately relaxed is shown in Figure 13. A knitted image made using this pattern is shown in Figure 1. After knitting this image, the floats that have emerged in the area where constraints were relaxed, were fixed manually. A sequence of images a user might generate while designing a knitted image based on a movie still is shown at Figure 15.

6 Evaluation

When seen at a sufficient distance the knitted pixel falls below the threshold of visual acuity, like a monitor pixel. We compare the similarity of the knitted patterns to the original drawings by simulating distance with a Gaussian blur. For a dense line-drawing (Figure 11) the best result was generated by the hybrid method, image (e). If the starting assignment had been more consistent, GSAT alone may have been sufficient. Based on empirical observation a starting assignment should have at least 70% of its variables consistently assigned for GSAT alone be considered. An image with large uniform areas and gradients (Figure 12) processed with an invasive ordered dithering (e) and then GSAT (f) creates an evenly shaded, grayed-out image, while using GSAT with a less invasive dithering (j,c) results in prominent alterations. A good reason to use ordered dithering is that a well-chosen dithering matrix produces lattice artifacts (Figure 4b) that partially satisfy the constraints. Because the artifacts consist of disconnected dots rather than lines they are are not visually jarring and the information they add to the resulting design is small [Arnheim 1974, p. 29]. An image processed with Atkinson dithering, which emphasizes the white or black areas, and pseudo-random texturing (i) is also uniformly shaded but lighter. Image (i) resembles the original the best, while image (h) best illustrates the qualities of a knitted image emerging from a distance. Turning off yarn anchoring increases image resemblance to the original and contrast (d). The desirability of such effects depends on the colors of the yarn and the content of the starting image.Although Min-Conflict and Random Walk produce unsatisfactory results when used alone, images (b,d,g), optimized with a hybrid method, exhibit fewer artifacts. Comparing figures 11,12,14, 15 we can see that the similarity between an initial pixel assignment and a knitted pattern grows with size of image. For large enough patterns starting assignments and knitted images differ only in brightness.



(a) *GSAT, flips:* 2694.



(b) *Pseudo-random texturing, flips:* 3566.

Figure 10: *Comparing GSAT and pseudo-random texturing on image 4c, $f = 4$.*

7 Discussion and Future Work

In a typical use-case, after an automatic optimization is run, a user may want to make some structure-restoring adjustments manually (Figure 13). While knitting the example image (Figure 1), despite having a knitting pattern of a satisfying visual similarity, we decided to intentionally relax constraints on the girl's face. Making such exceptions means that floats have to be fixed manually, which may be hard for a novice user. Sometimes artists modify a pattern to make reattaching yarn easier, and use a process called backladdering [Salomone 2013]. Back-laddered knitted fabric is still fragile, and therefore such exceptions should be used judiciously. Currently we assume that the impact of every variable flip that does not otherwise create a visible artifact is the same, but such a weighting is perceptually correct only for abstract images. In the future we intend to detect and handle different areas of the image differently. Certain areas of an image, for example anthropomorphic elements, receive more visual attention than others and it makes sense to weaken constraints where it contributes to the visual appeal and meaning of the design. In future we will detect important areas automatically using shape extraction, face detection and allow user intervention. This information can make constraints context-dependent.

In a different approach constraints may be treated as costs. The critical value of f in (2) may be interpreted as an interval cost constraint penalizing long floats. Adding a modification cost constraint would ensure that as few modifications are made as possible, and the CSP could be solved as a cost minimization problem.

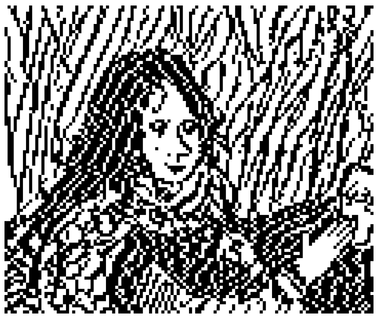


Figure 13: *A finished optimized and user-adjusted pattern. After the image in Figure 3a was optimized by GSAT, constraints in meaningful areas were deliberately relaxed by the user.*

The initial pixel assignment in our implementation does not take the artifacts of pixelation and scaling [Gerstner et al. 2013] into account. We could take additional steps to minimize artifacts and thus start with a better initial assignment. For example dithering might be improved using edge enhancement [Eschbach and Knox 1991]. Some yarns when knitted create non-square stitch pixels and in those cases a knitting pattern must be adjusted using yarn gauge information. We also plan to allow arbitrary image rotation when selecting the starting assignment. In images with many directional elements, such as twigs(Figure 3a), water ripples, or cross hatching, a designer might reinforce them by aligning them with the direction of knitting stitches. Knitting is not rotation-invariant, which provides an opportunity for interplay between knitted texture and image texture. As long as the starting image is large, we could sort structural areas of the image by brightness and tile them with a constraint-consistent texture patch of varied grayness. Such textures could be selected from a library [Salisbury et al. 1994], procedurally generated from patterns such as hair, swamp, grass, etc. [Lefebvre and Neyret 2003], or randomly [Lagae et al. 2010]. Building on the method described by Colton [Colton 2008], the user could mark areas of the source image as texture, from which a patch of constraint-consistent textured could be extracted and used to infer image-dependent constraints. The inferred rules are then reproduced to generate a consistent pattern for the whole area. In a qualitatively different approach, a pattern can be interpreted as image filtering, where an image filter over the source simultaneously scales down and discretized an image into a knitting pattern. Pairs of initial sketches and the corresponding manually-designed knitting patterns can create a filter that processes the image. Such an approach was applied [Colton and Torres 2009] to a colored image using evolutionary algorithms.

8 Conclusions

This paper has described and evaluated local search, texturing, and dithering for converting drawings or photographs to knitting instructions, capable of driving either a knitting machine or manual knitting. As a part of the evaluation we have implemented an open-source software prototype for automatically generating knitting patterns using a user-configurable combination of these algorithms, different for line-drawings and photographic source images, and manual pixel editing. Knitting patterns produced by the software were compared to the original images by viewing them at a distance. They exhibit a satisfying level of visual similarity. Use-case scenarios illustrating the use of the described approach have been discussed. One pattern produced by the software was knitted (Figure 1) and the results are satisfactory. Other patterns produced by the same technique result in images that are sufficiently similar to the originals and are neat at the back.

Appendix A

Online Resources:

- Brother Knitting machine data format
  `http://wiki.forskningsavd.se/ElectroknitKH940`
- Tutorial: Programming a Brother knitting machine
  `http://sternlab.org/2010/11/hacking-the-brother-kh-930e-knitting-machine/`
- Software used in this paper
  `http://www.cgl.uwaterloo.ca/~mkryven/knit.html`
- Andrew Salomone demonstrates using a knitting machine to print digital images at World Maker Faire, New York, 2011
  `http://youtu.be/iiTTrT29HI0`

References

ARNHEIM, R. 1974. *Art and Visual Perception: A Psychology of the Creative Eye*, 2nd ed. University of California Press.

BAYER, B. 1973. An optimal method for two-level rendition of continuous-tone pictures. In *Proceedings of the IEEE International Conference on Communications*, vol. 1, 11–15.

COLTON, S., AND FERRER, B. P. 2012. No photos harmed/growing paths from seed: an exhibition. In *Proceedings of the symposium on Non-Photorealistic Animation and Rendering*, 1–10.

COLTON, S., AND TORRES, P. 2009. Evolving approximate image filters. In *Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, 467–477.

COLTON, S. 2008. Experiments in constraint-based automated scene generation. In *Proceedings of the 5th international workshop on Computational Creativity*.

CUTTING, J. E. 2002. Representing motion in a static image: constraints and parallels in art, science, and popular culture. *Perception 31*, 10, 1165–1193.

ESCHBACH, R., AND KNOX, K. T. 1991. Error-diffusion algorithm with edge enhancement. *J. Opt. Soc. Am. A 8*, 12, 1844–1850.

FLOYD, R. W., AND STEINBERG, L. 1976. An Adaptive Algorithm for Spatial Greyscale. *Proceedings of the Society for Information Display 17*, 2, 75–77.

GERSTNER, T., DECARLO, D., ALEXA, M., FINKELSTEIN, A., GINGOLD, Y., AND NEALEN, A. 2013. Pixelated image abstraction with integrated user constraints. *Computers & Graphics 37*, 5 (8), 333–347.

GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. X. 2004. Programmable style for NPR line drawing. In *Proceedings of the 15th Eurographics conference on Rendering Techniques*, 33–44.

GRISWOLD, R. E. 2007. Mathematical and Computational Topics in Weaving. `http://www.cs.arizona.edu/patterns/weaving/webdocs/mo.pdf` (accessed May 2013).

HOOS, H. H., AND STÜTZLE, T. 2000. Local Search Algorithms for SAT: An Empirical Evaluation. *J. Autom. Reason., 24*, 4, 421–481.

IGARASHI, Y., IGARASHI, T., AND SUZUKI, H. 2008. Knitting a 3D model. *Computer Graphics Forum, 27*, 7, 1737–1743.

IGARASHI, Y., IGARASHI, T., AND MITANI, J. 2012. Beady: interactive beadwork design and construction. *ACM Trans. Graph., 31*, 4, 49:1–49:9.

IJIRI, T., OWADA, S., OKABE, M., AND IGARASHI, T. 2005. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. *ACM Trans. Graph., 24*, 3, 720–726.

JODOIN, P.-M., EPSTEIN, E., GRANGER-PICHÉ, M., AND OSTROMOUKHOV, V. 2002. Hatching by example: a statistical approach. In *Proceedings of the symposium on Non-Photorealistic Animation and Rendering*, 29–36.

LAGAE, A., LEFEBVRE, S., COOK, R., DEROSE, T., DRETTAKIS, G., EBERT, D., LEWIS, J., PERLIN, K., AND ZWICKER, M. 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum, 29*, 8, 2579–2600.

LEFEBVRE, S., AND NEYRET, F. 2003. Pattern based procedural textures. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, 203–212.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Trans. Graph. 26*, 3.

OSTROMOUKHOV, V. 2013. Non-photorealistic Shading and Hatching. In *Image and Video-Based Artistic Stylisation*, vol. 42 of *Computational Imaging and Vision*. Springer London.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc.

RADCLIFFE, M. 2008. *The Essential Guide to Color Knitting Techniques*. Storey Publishing, LLC.

RUSSELL, S. J., AND NORVIG, P. 2009. *Artificial Intelligence: A Modern Approach*, 3 ed. Pearson Education.

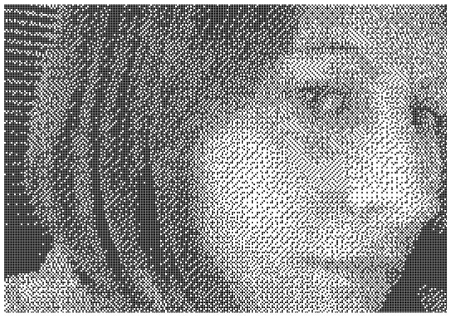
SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 101–108.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 401–406.

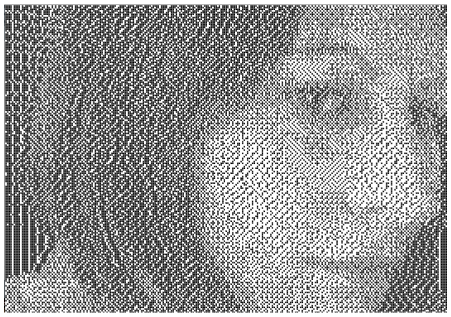
SALOMONE, A., 2013. Personal communication, May 18, 2013.

SINGH, M., AND SCHAEFER, S. 2010. Suggestive hatching. In *Proceedings of the Sixth international conference on Computational Aesthetics in Graphics, Visualization and Imaging*, 25–32.

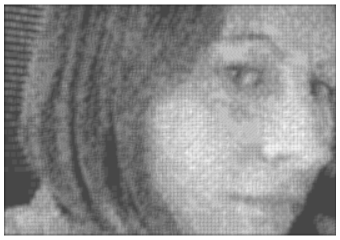
YUKSEL, C., KALDOR, J. M., JAMES, D. L., AND MARSCHNER, S. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph. 31*, 4, 37:1–37:12.



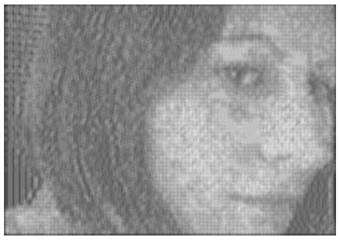
(a) *Initial assignment generated using Floyd-Steinberg dithering. Total conflicts remaining: 16741*



(b) *A knitting pattern optimized with GSAT, f=4. Number of flips:2846*

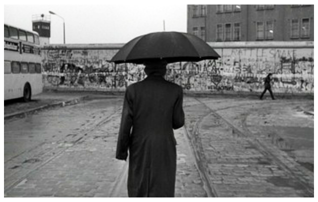


(c) *Initial Floyd-Steinberg assignment as seen from a distance.*



(d) *A GSAT knitted pattern as seen from a distance.*

Figure 14: *Image size: 267x186, a photograph taken with a web-camera.*

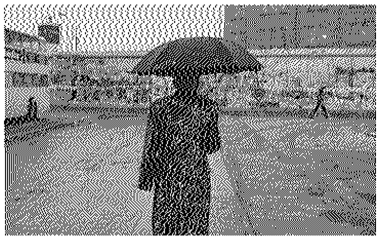


(a) *The original photograph.*

(b) *Original photograph seen from a distance.*

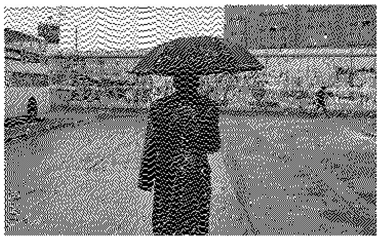
(c) *Floyd-Steinberg dithering starting assignment.*

(d) *Starting assignment from a distance.*

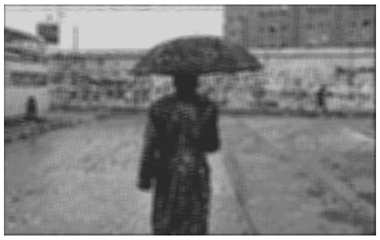


(e) *A knitting pattern, knitting in vertical direction.*

(f) *Vertical knitted pattern seen from a distance.*



(g) *A knitting pattern, knitting in horizontal direction.*



,

(h) *Horizontal knitted pattern seen from a distance.*

Figure 15: *Use-case: designing a knitted scarf based on a movie still taken from Wim Wenders, Wings of Desire. Image size:* 363×255.