Constraint Solving for Inkjet Print Mask Design

Jonathan Yen*
Hewlett Packard Laboratories, Palo Alto, California

Mats Carlsson
Swedish Institute of Computer Science, Sweden

Michael Chang†
Hewlett Packard, San Diego, California

Joan Manel Garcia
Hewlett Packard, Barcelona, Spain

Hugh Nguyen‡
Hewlett Packard, Cupertino, California

We present results in a constraint solving approach for automatic generation of Inkjet print masks. Print masks are used to control the firing of the nozzles, that is, to determine which nozzles on an Inkjet printer cartridge are to spit an ink droplet at each particular instant in a multiple-pass print mode. Many design rules for print masks can be modeled in terms of constraints and cost functions. For example, if adjacent nozzles are fired simultaneously, printing artifacts often result. Therefore, spatial adjacency constraints with respect to horizontal, vertical and diagonal neighbors are modeled with various cost functions. Minimizing the associated, total costs then generates the print masks. Initial solutions are found by a greedy algorithm with some randomization; then neighborhood search techniques are applied to find local near-optima. Our approach can generate masks for Inkjet printers in multiple-pass print modes for multiple-level, multiple-drop technologies. It has been used to help design the print masks for Hewlett Packard's wide format printers (DeskJet 2500C and 2500CM). This approach can shorten the turnaround time for print mask design in a systematic and methodical way.

Journal of Imaging Science and Technology 44: 391–397 (2000)

Introduction

In the thermal inkjet printer technology, an inkjet printer cartridge, an inkjet pen, contains arrays of nozzles such that an ink droplet is spit from each nozzle when a micro explosion occurs if the ink chamber is heated. Print masks are used to control the firing of the nozzles, i.e., to determine whether one particular nozzle is to spit an ink droplet at one particular instant.

There are various kinds of print mode in thermal inkjet printing technology. The most basic one is the single pass, single drop print mode. The pen sweeps across the media and puts a drop on the pixel if and only if there is a data point at the pixel location that needs to be marked. Afterwards the media is advanced exactly the pen length. Therefore, there is only one chance that the pen visits each pixel location. Furthermore, there is at most one drop to be put on the media per visit. Therefore, it is a single level printing. In this type of print modes, there is no need for extra control logic to control the firing of the nozzles.

However, many things can go wrong. For example, the media advance error can produce misalignment in the output. Ink coalescence can be an unpleasant artifact that is caused by the fact that ink may migrate before it is completely dried. Firing the nozzles at higher frequency than they can handle may cause puddling around the nozzle opening that can create the satellites. Multiple-pass print modes are usually used to avoid these printing artifacts. In a multiple pass print mode, the pen visits each pixel location more than once. Therefore, extra logic control is required to determine whether the pen should put a drop in this pass when this pixel location is to be marked. Print masks are applied to provide this extra logic. They are usually coded in an array of 0's and 1's. 1 indicates firing the nozzle if there is a data point needs to be marked. 0 means no firing regardless if there is a

Original manuscript received March 15, 2000

* Corresponding author

† Present address: Peerless Systems Corporation, San Diego, California

‡ Present address: AltaVista Company, Mountain View, California

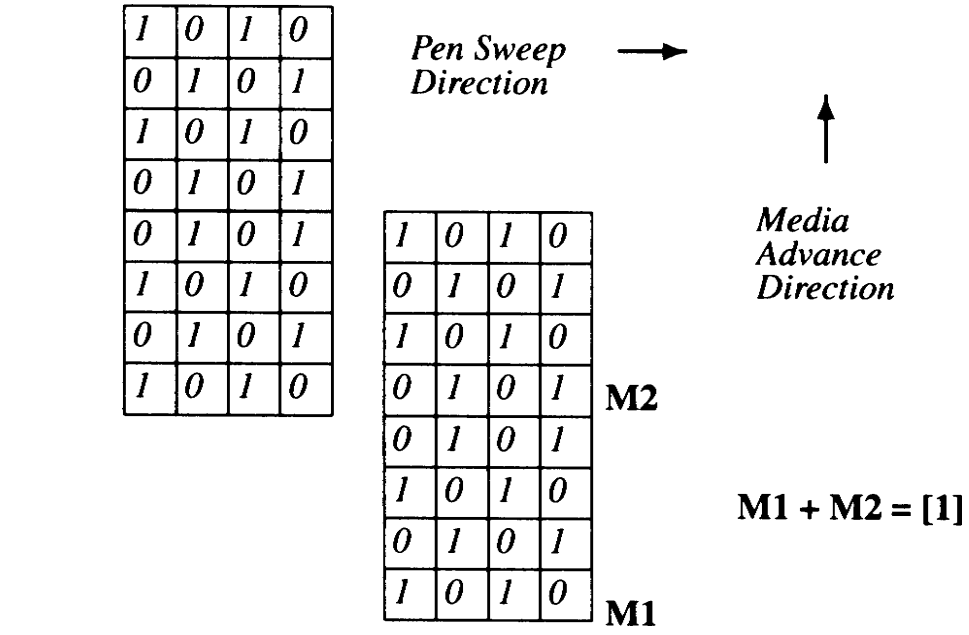

Figure 1. A two-pass print mask

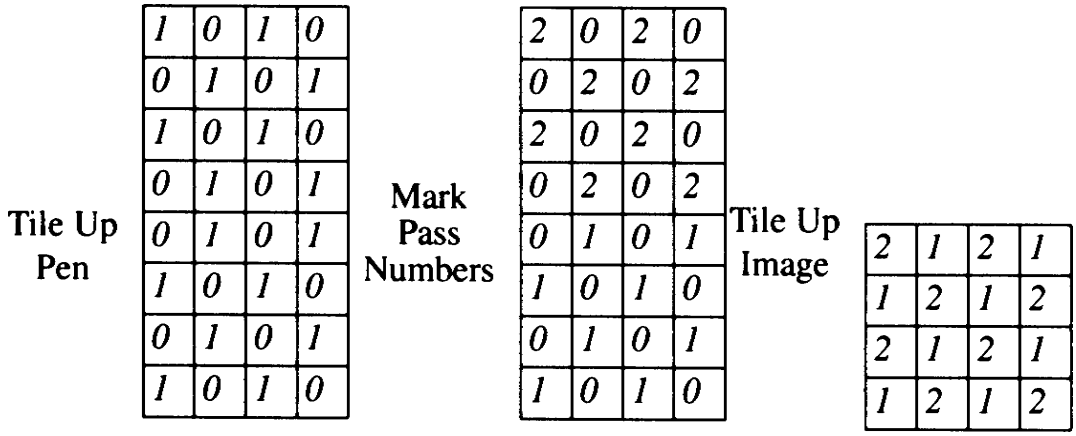

Figure 2. Problem formalization

data point. The size of the array varies from one pen architecture and writing system to another. The arrays tile up to cover the printable area of the media. Each array element has direct correspondence to a particular pen nozzle in a particular pass, controlling the firing as the column of pen nozzles sweeps across the media.

Figure 1 shows an example in a two-pass print mode, with print mask width four and length eight. As this mask array is tiled over the media, the pen sweeps across the media and marks every other pixel location if there is a data point there, in a checkerboard pattern. This alternating fashion is to avoid consecutive firing of the same nozzle. Afterwards the media is advanced half of the pen length and the printing resumes. Notice if we label the lower half of the array as *M1* and the upper half *M2*, then *M1* and *M2* should be complementary to each other. In other words, all the entries that are missed by *M1* the first time when the pen sweeps along are picked up by *M2* during the second time when the pen visits again.

The print masks are important because they directly affect the print quality. They provide the necessary logic control in multiple-pass print modes. They enforce hardware limitations on firing frequency. This can be modeled as a spatial adjacency problem. In case one wants to further avoid the adjacency between different passes, then it is a temporal adjacency problem. If there is a defective nozzle, one can shift its printing responsibility to another nozzle by specifying exactly that in the print masks. More aggressively, we have proposed to apply halftone patterns for print masks.[1] By introducing an invisible halftone pattern or a less objectionable pattern, one can disrupt the unpleasant, existing pattern, or cover up by the new pattern.

Problem Formulation

Mask arrays are used to tile up the entire printable area on the media. Because the logic in 1 means firing at this pixel location when the pen sweeps across the media, we could have marked the non-zero entries by

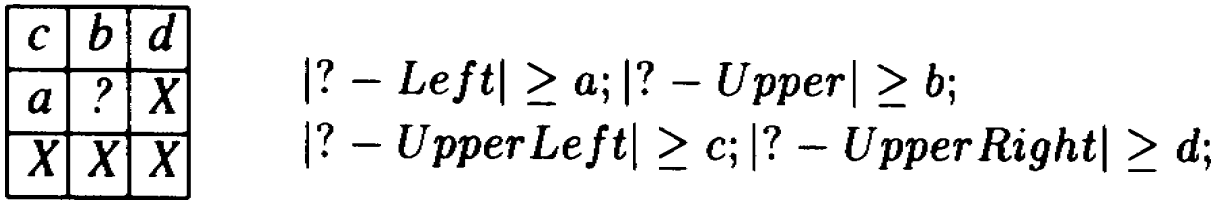

Figure 3. Constraint Matrix

Level 1: 1 drop
Level 2: 3 drops

2 247	6 356	4 124	8 358
6 368	2 127	8 568	4 147
4 124	8 358	2 247	6 356
8 568	4 147	6 368	2 127
6 368	2 127	8 568	4 147
2 247	6 356	4 124	8 358
8 568	4 147	6 368	2 127
4 124	8 358	2 247	6 356

Figure 4. A multi-level example

the *pass number* instead of 1. Looking from the pixel's point of view, these two have one-to-one correspondence. Furthermore, we could combine the upper and lower into one because they are complementary. And these two have one-to-one correspondence. This representation is more compact and yet complete (Fig. 2).

Now the task for mask generation is to fill up the mask array with numbers subject to the adjacency constraint. The notion of a Constraint Matrix was invented for this purpose (Fig. 3).[2] Each time when we are to decide the number to be filled in this entry, we know that it has to be at least ***a*** distance away from its left neighbor, ***b*** distance from its upper neighbor and ***c*** distance from its upper left neighbor, and so on; ***x***'s are don't-care's. This representation is simple and effective. The masks generated by this method have been used in HP's large format printers.

All these have worked well in single-level printing. Now we have to extend it to handle multiple-level print modes. The difference here is that in single level printing, among multiple visits of the pen over the media, on one, and only one of the visits will the pen put a drop on the pixel location. However, with diluted ink, or smaller drop volume, one could put a drop for more than one of the visits during multiple passes.

Here is a practical example (Fig. 4). We conduct our feasibility test on one of HP's prototype printers that uses the 64-nozzle color pen. The print mask has length 8 and width 4. In one particular print mode, it is 8-pass, with 3 levels. For example, if the image data at the pixel location corresponding to the first entry is level 1, then the pen puts one drop, in pass 2. If the image data here is level 2, then the pen puts three drops, one in pass 2, one in pass 4 and one drop in pass 7. If the image data here is level 3, then the pen puts eight drops, one drop for each pass (there is no need for logic control in level 3).

Constraint Modeling

The entry of the mask array is extended to a cell (Fig. 5). In the previous example, all the pass number appears in the top row in level 1 are always included in the second row in level 2. This dependency property is not unusual in many practical cases. It is also possible that same pass number may appear more than once within a row. The repetition of the same pass number actually means more than one drop can be fired at each

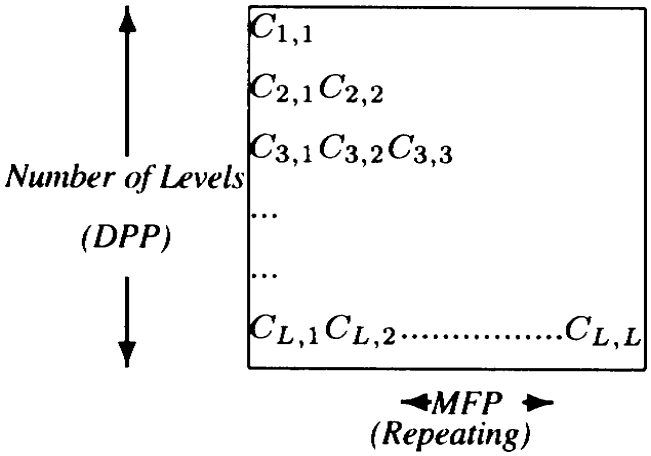

Figure 5. A cell

pixel location per pass. This is called the multiple-pass, multiple-drop, multiple-level print mode.

Therefore, a cell is a structure such that there are at most DPP rows, where DPP stands for drop-per-pixel. Each row is a bag, a collection of pass numbers in which some of them may repeat up to MFP times (Maximal Firing Per pixel per pass). The dependency in the previous example can be achieved by assigning portion of the solution as input to the constraint solver.

The total cost will be the composite costs of the adjacency constraints, the evenness of the mask, and the halftone coherence. The cost of the adjacency constraints is the sum of the costs of the constraints that are false in the mask. The evenness encourages the number of occurrences of values to be as even as possible. The cost associated with the halftone coherence is intended to disrupt the halftone pattern in the output.

We now have a list of the input parameters. P is the number of passes. The dimension of the mask array is the pen length divided by P. The width is a user input, dependent on the particular writing system. It can be a two-dimension mask array, one for each particular color. It can also be layers of mask arrays such that we may or may not want to enforce constraint across the layers. The DPP, drop per pixel, corresponds to the number of levels. The MFP is the maximal firing per pass per pixel. A set of weight tables defines the costs. A partial assignment (maybe a halftone pattern) to the solution may also be specified.

The solution is a mask array of given dimension such that each entry is a cell. A cell contains DPP bags of different size. Each slot of the bag is assigned an integer between 1 and P such that there are MFP occurrences of any integer in the bag. The mask array is a near-optimal solution to the global minimization of the total cost.

Constraint Solving

The constraint solving strategy is based on a greedy, randomized, adaptive search procedure (GRASP)[3] followed by a repair phase. The greedy phase computes a solution by filling in one cell at a time while minimizing the accumulated cost. Alternative values are chosen with a probability that is proportional to their benefit to the total solution. The user can tune the degree of randomness.

The repair phase uses the solution computed by the greedy phase as a starting point for a neighborhood search. We have currently implemented a simple hill-climber that stops when it has found a solution with locally minimal cost, but other options are possible, e.g. simulated annealing or taboo search.

By repeating the greedy and repair phases, the constraint solver generates a stream of proposed, locally optimal solutions. An interesting idea is to use the greedy phase to produce an initial population of solutions, and then use a genetic algorithm to let the population evolve towards better fitness values.

Details on the constraint modeling and the constraint solving are described in Appendix A.

The Results

We have implemented a constraint solver, in TCL/TK, a script language, as the input interface, and a text file as the output.

An input script file in TCL can be found in Appendix B. The programmable support routines are included in the file "support.tcl". A random seed is generated for each instance. By recording this seed and setting the random seed accordingly, one can repeatedly generate the same mask array. This example is a spatial adjacency problem (SAP) with a mask array, of width 4 and height 8, one layer (one color plane), for an 8-pass, 3 drops-per-pixel (3-level), single-drop technology. The evenness is in full weight (1.0) and the attenuation about the halftone coherence is set to be 50%. The mask array should be considered as wrapped-around in all directions because, for example, its top most row is actually immediately adjacent to its bottom most row when it is used to tile up the image. The default cost is ranged from 0 and 15. No horizontal adjacency is allowed, i.e., the cost is infinity if the left neighbor is the same as the current entry. Vertical or diagonal adjacencies are allowed, but not without some penalty. Some GRASP selections can be specified also, for instance, greedy search and hill climber, and so on. A sample output can be found in Appendix C. Sometimes one has to translate the mask array into an internal hexadecimal representation that is specific to the prototype printers to be tested. A number of enhancements and extensions have been made due to the preliminary implementation. For example, a newer version of our constraint solver handles a special variation, i.e., using print masks in 300 dot-per-inch (dpi) resolution for printing 600 dpi data by extending each pixel into four quadrants.

Summary

Print masks are always among the last to be finalized before the manufacturing release. They are always under severe limitations. However, miracles are always expected to fix problems at the last moment and miracles are always expected in a very short time. The main objective of this work is to shorten the turn around time for print mask design.

Not all of the desirable characteristics can be modeled as constraints and cost functions easily, if at all. There are still too many possibilities to test. A strategy is needed to produce a collection of representative masks within the solution space. It is fully understood that there will always be inconsistency across pens and across different printers. Therefore, even optimal solutions are never guaranteed to produce the best print quality.

More and more complicated pen architecture will be introduced. The expected life span of pens will be much longer with the new, off-axis-ink technology. Manufacturing cost cutting will end up with less expensive and

less accurate parts such that error hiding will be more and more important. Therefore there is a need for an automated tool for print mask design. We believe that constraint solving suits this type of problems.

Appendix A: A Spatial Adjacency Problem

Constraint Model

In its basic form, the problem consists of finding an array A such that each entry $A[ij]$ has an integer value in $1..p$, $p \leq 32$, subject to a number of disequality constraints $A[ij] \neq A[i'j']$.

There are three extensions to the basic problem specification:

Soft Constraints

Each constraint is associated with a cost that is a floating-point number in $[0, \infty]$. 0 denotes that the constraint is disabled; ∞ denotes that it is mandatory. The total cost of an array A is defined as the sum of the costs of the constraints that are false in A. Furthermore, it is desirable to have the number of occurrences of values in $1..p$ as even as possible; deviations from the mean value contribute to the cost function too. The problem thus becomes a Constraint Satisfaction Optimization Problem (CSOP) whose objective is to find an array with minimal cost, subject to the mandatory constraints.

Three-Dimensional Constraints

The array A is generalized to be three-dimensional.

Composite Constraints

Each entry of the array A is extended into a cell containing some number of bags of different sizes $s_1 < \ldots < s_n$. All cells have the same number of bags of the same sizes. A bag of size s_i is said to be at level i.

Each slot in a bag should be assigned an integer in $1..p$, such that there are at most mfp occurrences of any integer in the bag, where mfp is a problem parameter.

A constraint on two cells A_{xyz} and $A_{x'y'z'}$ with weight w is interpreted as the conjunction of the following, where $b \cong 0.5$ is a problem parameter:

- All disequalities with weight w between some slot at level i of A_{xyz} and some slot at level i of $A_{x'y'z'}$.
- All disequalities with weight bw between some slot at level i of A_{xyz} and some slot at level $i - 1$ of $A_{x'y'z'}$.
- All disequalities with weight bw between some slot at level i of A_{xyz} and some slot at level $i + 1$ of $A_{x'y'z'}$.

Cost Function

Let $\#(v,A)$ denote the number of occurrences of a value v in the array A, let $\#(v,i,C)$ denote the number of occurrences of a value v at level i in cell C. Then the cost of a solution instance A is defined as the sum of the following terms:

- For all X coordinates x,x', Y coordinates y,y', Z coordinates z,z', levels i, and values v in $1..p$, such that (x,y,z) is lexicographically smaller than (x',y',z'):

$$bw \times \#(v,i,A_{xyz}) \times \#(v,i\text{-}1,A_{x'y'z'}) + w \times \#(v,i,A_{xyz}) \times \#(v,i,A_{x'y'z'}) + bw \times \#(v,i,A_{xyz}) \times \#(v,i+1,A_{x'y'z'})$$

where w is the weight associated with the constraint on A_{xyz} and $A_{x'y'z'}$.

- For all values v in $1..p$:

$$e \times | \#(v,A) - \lfloor S/p \rfloor |$$

where e is a problem parameter denoting the relative importance assigned to evenness and S is the total number of slots.

- For all X coordinates x,x', Y coordinates y,y', Z coordinates z,z', such that (x,y,z) is lexicographically smaller than (x',y',z'):

$$c \times \sum_i \sum_j P(i, j, A_{xyz}, A_{x'y'z'}) \times D(i, j, A_{xyz}, A_{x'y'z'})$$

where c is a problem parameter denoting the relative importance assigned to halftone coherence. Setting it to zero removes the halftone coherence completely from the cost function;

$P(i, j, A_{xyz}, A_{x'y'z'})$ is the probability that level i occurs in the halftoned images at the cell A_{xyz} and level j occurs at the cell $A_{x'y'z'}$.. Alternatively, we can replace this term by its reciprocal to disrupt the halftone patterns;

$D(i, j, A_{xyz}, A_{x'y'z'})$ is a function of the distance in pass number between level i in cell A_{xyz} and level j in cell $A_{x'y'z'}$. Let I be the set of all pass numbers to print level i in cell A_{xyz} and J be the set of all pass numbers to print level j in cell $A_{x'y'z'}$, then the distance in pass number is defined to be

$$min_{\,u \text{ in } I,\, v \text{ in } J} \{ |u\text{-}v| \}.$$

Partial Assignments

For selected slots, a value that must not be changed by the algorithm, or a set of values that must not be assigned by the algorithm, can be chosen.

Problem Parameters

Summarizing, a problem instance is completely defined by the following parameters:

- an integer p defining the domain of all slots,
- three integers D_x, D_y, D_z defining the size of the three-dimensional array,
- a set of disequality constraints, each specified as two array coordinates and a weight, i.e., as a triple

$$<(x,y,z), (x',y',z'), w>$$

where w is a floating-point number,

- an integer DPP, and some integers $s_1 < \ldots < s_n$ = DPP denoting the relevant bag sizes and levels,
- an integer MFP, denoting the maximum number of occurrences of any integer in any bag,
- a floating-point number b denoting the attenuation factor in adjacent layer constraints,
- a floating-point number e denoting the evenness contribution to the cost function, any pre-assigned or forbidden values for selected slots

Convenience. For convenience, a number of alternative ways of specifying the input constraints have been

defined. Let the three Boolean parameters T_x, T_y, T_z denote whether the array "wraps around" the respective dimension in the following description:

Repetitive Constraints

These are interpreted as follows: For each cell c, a constraint between c and another cell located at a fixed 3-D offset from c. The treatment of cells near the borders depends on the tiling parameters.

A repetitive constraint is defined as a tuple

$$<(\delta_x, \delta_y, \delta_z), (l,u)>$$

where δ_x, δ_y, δ_z are integer offsets and $l \le u$ are floating-point weights denoting that the weight of each individual constraint shall be a random number in $[l,u]$.

Specific Constraints

These are specific constraints on coordinate pairs.

A specific constraint is defined as an triple

$$< (x,y,z), (x',y',z'), (l,u) >$$

where *(x,y,z)* and *(x',y',z')* denote two matrix locations and *l, u* are as above.

Default Constraints

These are interpreted as follows: For each pair of cells that is not otherwise constrained, a constraint may be imposed whose weight depends on the distance between the cells.

A default constraint is defined as a pair (l,u). The weight of each constraint is defined as r/d where r is a randomly chosen number in $[l,u]$ as described above and d is the distance between the two cells, taking wrap-around into account.

The offset in any dimension between two indices i and i', with array size s along the chosen dimension, is defined as:

$\lvert i-i' \rvert$	without tiling, or
$min(\lvert i-i' \rvert, s-\lvert i-i' \rvert)$	with tiling.

The distance between two matrix locations at a 3-D offset δ_x, δ_y, δ_z is defined as:

$$\sqrt{\delta_x^2 + \delta_y^2 + \delta_z^2}$$

Implementation

The implementation consists of three parts:

- An initialization module that initializes the matrix, sets up the constraints and computes initial solution(s), using a greedy algorithm.
- An optimization module that improves the initial solution(s), using neighborhood search methods and/or a genetic algorithm.
- A harness module whose task it is to perform I/O and to combine the other modules.

Requesting specific heuristics can control each module.

The Initialization Module

The greedy algorithm computes an initial solution with reasonably low cost. The algorithm traverses the matrix, assigning one cell slot at a time. For each cell slot, the algorithm chooses the value that minimizes the cost so far, combined with an element of random choice so as to provide randomized solution sample for input to the optimization module.

Two parameters G_c and G_r in $[0, \infty]$ are introduced, denoting the relative importance of these heuristics. The greedy algorithm chooses for cell slot x the value v that minimizes:

$$(\delta_{cost}(v) + G_c) * (\text{random}(v) + G_r)$$

where $\delta_{cost}(v)$ is the contribution to the cost function if x is assigned to v, and *random(v)* is a random number in [0.0,1.0]. For example, $G_c = G_r = 0$ yields a choice of value with probability proportional to $\delta_{cost}(v)$; $G_c = G_r > 0$ makes the choice depend almost entirely on $\delta_{cost}(v)$; $G_c > 0$ $G_r = 0$ makes the choice almost random.

The Optimization Module

Starting from the solution produced by the greedy algorithm, this module performs a neighborhood search for better solutions until a local cost minimum is found.

The Harness Module

This module provides an iterative randomized sampling of solutions by repeatedly invoking the initialization and optimization modules.

Appendix B: A Sample TCL Script

```
#!/bin/sh
# Switch to Tcl/Tk \
exec ./hpcs -f $0

#supporting stuff in tcl
source support.tcl

#find a random number
set random [expr int(rand( ) * 100000000)]
SetRandomSeed $random #set the random seed

# Create a problem instance. Syntax:
# SAP <name> <width> <height> <depth>
# <#passes> <list of levels> <mfp>
# <evenliness> <attenuation>
# <wrapx?> <wrapy?> <wrapz?>
SAP sap 4 8 1 8 {3} 1 1.0 0.5 yes yes yes

#set a range for the default cost
sap default 0 15
# forbid horizontal adjacencies:
sap repetetive  –1   0   0   inf
# avoid other adjacencies:
sap repetetive  –1   –1   0   1   5
sap repetetive   0   –1   0   3   10
sap repetetive  –1    1   0   1   5

# Create a printmask
Printmask printmask sap
Greedysearch printmask
Hillclimber printmask
```

```
# Print output
set output [printmask get]
PrintMatrix $output$

Exit
```

Appendix C: A Sample Output Mask

8	2	4	5
148	257	148	357
3	2	5	6
357	246	357	168
6	2	3	5
367	245	136	258
1	2	7	3
167	248	167	358
4	3	6	7
468	357	168	237
8	4	1	2
138	247	158	246
4	3	1	6
247	356	124	368
6	1	4	5
346	125	347	158

References

1. J. Yen, Q. Lin and P. Wong, Improved Print Masks for Inkjet Printers, *U.S. Patent* 5,992,262 (1999).
2. J. M. Garcia, *On the Design of Print Masks,* Internal Report, IJBU-Barcelona, Hewlett-Packard Co., 1997.
3. T. A. Feo and M. G. C. Resende, Greedy Randomized Adaptive Search Procedure, *J. Global Optimization*, **6,** 109–133 (1995).