Algorithmic Part for Printing-Based Microfabrication

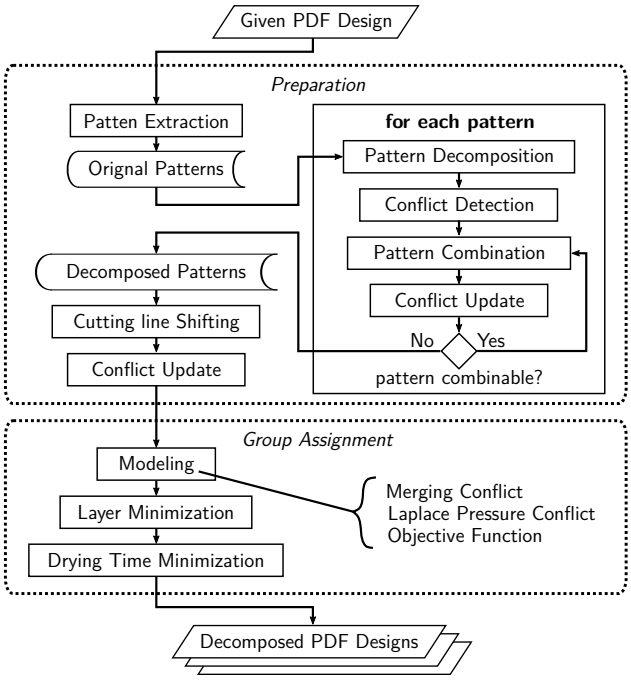


Figure 1: The overall flow of the proposed algorithm.

1 Algorithm and Software

The core idea of our algorithm is to decompose the entire design into groups of small patterns and then print them separately to avoid potential printing problems. Specifically, patterns assigned to the same group are guaranteed to cause no printing problem, so that by serially printing each group, i.e., printing a group only after the solvent for printing its previous group evaporates completely, patterns are safely superposed to achieve the target design.

The algorithm in general consists of two major steps: (1) the preparation step (Section 1.1) and (2) the group assignment step (Section 1.2). Figure 1 shows the overall flow of the proposed algorithm. We have developed a software tool to implement the algorithm. The tool takes a PDF file containing the target design as its input and outputs several PDF files as the grouping results.

1.1 Preparation

The preparation step further consists of several sub-steps: With the given PDF design, we perform **pattern extraction** to extract the original patterns, each of which is a closed drawing path. For each original pattern, we first perform **pattern decomposition** to decompose the original pattern into small patterns. Then we perform **conflict detection** to locate the hotspots for merging and Laplace pressure problems. We define a conflict to occur when a pre-defined threshold, e.g., the spacing distance or the dimension difference of patterns, is violated. A conflict may not necessarily lead to a printing problem, but the printing result cannot be guaranteed. After conflict detection, we perform **pattern combination** and then **conflict update** to combine some decomposed small patterns back to large patterns. In this way, the pattern number can be reduced to save some computational effort required for the later group assignment step. Finally, to ensure that contacted patterns will be merged together but not separate after printing, we perform **cutting line shifting** to create an overlapping region for the contacted patterns, and one more conflict update is followed to specify new conflict relations that may be caused by cutting line shifting.

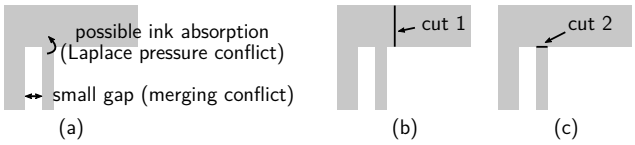


Figure 2: An example of pattern decomposition by cutting the pattern. (a) The original pattern with two intra-conflicts. (b) A decomposition result that resolves no intra-conflict. (b) A decomposition result that resolves both intra-conflicts.

All the information obtained in this step including pattern area, pattern drawing path, spacing distances between patterns, and conflict relations between patterns, is sent to the group assignment step as the input for optimization.

Except pattern extraction, which is straightforward implemented by extracting the path information from the given PDF file, the algorithms applied in all sub-steps are described as follows.

1.1.1 Pattern Decomposition

Though conflicts occuring between different patterns (*inter-conflicts*), can be resolved by group assignment, conflicts occuring within the same pattern (*intra-conflicts*) are not resolvable directly by group assignment. Figure 2(a) shows a pattern with two intra-conflicts. To resolve the intra-conflict, we propose to decompose the pattern containing

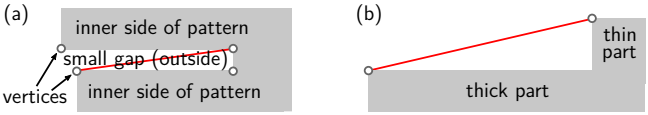


Figure 3: Deduction of concave polygons for (a) merging conflict (b) Laplace pressure conflict.

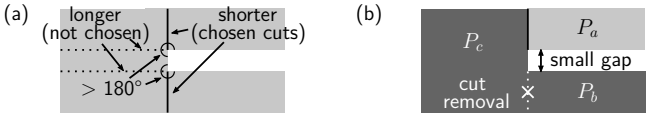


Figure 4: (a) Cutline selection in design decomposition. (b) Conflict inheritance in pattern combination.

the intra-conflict and assign the conflict parts to different small patterns, i.e., transforming the intra-conflict into an *inter-conflict*.

However, there are many ways to decompose patterns, and an arbitrary decomposition method may fail to resolve the intra-conflict. Figure 2(b) and (c) show two examples of pattern decomposition, both decomposing the pattern shown in Figure 2(a) into two small patterns, but the outcomes are much different.

In general, a smaller pattern is less possible to have the intra-conflict, but decomposing the original pattern into relatively smaller patterns indicates the increase of pattern numbers, which can cause a computational burden for the later group assignment. Therefore, the target of the proposed decomposition method is to resolve all intra-conflicts, while avoiding performing redundant decomposition.

By analyzing the formation of intra-conflict, we notice that if an intra-conflict occurs, it implies that the concerning pattern must be a concave polygon. That is, there exists at least one line connected to two vertices (a diagonal) of the polygon lies partially/completely outside the polygon. Figure 3 shows the analysis of the two conflict types, where the diagonals are colored in red. In other words, if the concerning pattern is a convex polygon, it is guaranteed that no intra-conflict can occur.

In our method, to achieve convex patterns, we first treat the original pattern as a polygon and traverse all its vertices. For each vertex whose corresponding internal angle is larger than 180 degree, we extend the two edges connected to the vertex toward the inside of the polygon until they reach the polygon boundary. The shorter extension is then applied as a cut to decompose the original design. An example is shown in Figure 4(a). By iteratively performing the cutting process, every internal angle that is larger than 180 degree will be divided into two angles: one is 180 degree and another is less than 180 degree, meeting the condition that all resultant polygons/patterns are convex.

1.1.2 Conflict Detection

After design decomposition, we obtain a set of convex patterns. In this sub-step, we specify the conflict relation between patterns.

To detect the merging conflict, for each pair of *non-contacted* patterns, we measure their shortest distance. Two patterns will be marked to have the merging conflict with each other if their spacing distance is smaller than a constant α. α is user-defined and has a default value as $\min(\rho, \delta)$, where ρ is determined according to the design size (multiplying the perimeter of the smallest bounding box encircling the entire design by 0.01), and δ is determined according to the pattern spacing distance (multiplying the minimum distance between any pair of patterns in the design by 3).

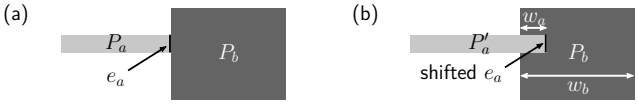


Figure 5: (a) Decomposed patterns before cutting line shifting. (b) Overlapping introduction after cutting line shifting.

To detect the Laplace pressure conflict, for each pair of *contacted* patterns, we compare their area and the lengths of their contacted edges. If their area ratio (the thin one over the thick one) is smaller than a constant β and the length ratio of their contacted edges (still the thin one over the thick one) is smaller than a constant γ, we mark these patterns to have the Laplace pressure conflict. Note that unlike the merging conflict that is a bidirectional relation, here only the thin pattern can be absorbed by the thick pattern. Similar to α, β and γ are also user-defined, and by default β is set to 0.2 and γ is set to $\sqrt{0.2}$.

1.1.3 Pattern Combination and Conflict Update

In this sub-step, we combine contacted patterns back to larger patterns to save the computational effort required for the later group assignment step. The patterns that are contacted and do not have any conflict relation are marked to be allowed for combination. For each time patterns are combined, their conflict relations will be inherited. Figure 4(b) shows an example, where there are initially three patterns P_a, P_b, and P_c. P_a has a merging conflict with P_b. P_c is contacted with both P_a and P_b and does not have the Laplace pressure conflict with either of them. If we combine P_b and P_c, the resultant pattern cannot be further combined with P_a, since it inherits the merging conflict between P_a and P_b.

1.1.4 Cutting Line Shifting

As metioned in Section 1.1.1, we cut the original patterns to realize pattern decomposition. Figure 5(a) shows two contacted patterns P_a and P_b that once belonged to the same original pattern. If we directly print P_a and P_b, it is possible that the printed patterns of P_a and P_b are separate but not merged together. Therefore, to ensure that the contacted patterns will not be actually "cut apart", we shift the shorter contacted edge, namely the *cutting line*, to produce an overlapping region. Figure 5(b) shows the shifting result of Figure 5(a), where e_a of P_a is shifted toward P_b to form a new P'_a. The shifting length (w_a in Figure 5(b)) is defined as the product of a constant ϵ and the depth of the unchanged pattern viewed from the cutting line (w_b in Figure 5(b)). ϵ, similar to other constants, is user-defined and has a default value as 0.2. However, if the resulting w_a is too small to achieve a sufficiently large overlapping region, i.e., $w_a < \rho$ (ρ is defined in Section 1.1.2), the default value of ϵ will be set to 0.7.

After cutting line shifting, some patterns are extended, leading to some new conflicts. Thus, we perform one last conflict update at the end of the preparation step. The whole preparation step is implemented using Python, and its outputs are stored as plain-text files.

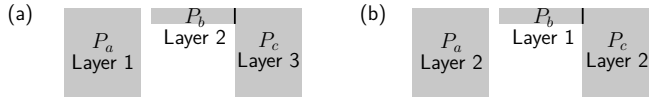


Figure 6: Different group assignment results. (a) Suboptimal. (b) Optimal.

1.2 Group Assignment

Since both types of printing conflicts can be resolved by assigning the conflict patterns to different groups, an intuitive but safe method is to label each pattern as an individual group. However, patterns can actually be printed together without needing to wait others to dry, as long as the patterns assigned to the same group have no conflict.

Nevertheless, choosing a pattern to print implies that all patterns conflictng with the chosen pattern become temporarily unprintable, and arbitrarily choosing patterns to print can lead to suboptimal solutions. Figure 6 shows an example, in which P_a has a merging conflict with P_b, and P_b (thin) has a Laplace pressure conflict with P_c (thick). If P_a is first chosen for printing, i.e., P_a is assigned to the first group, as shown in Figure 6(a), P_b cannot be printed together with P_a owing to the merging conflict, and P_c cannot be printed either since it can only be printed after P_b is printed. Eventually, three groups are needed if P_a is printed first. Actually, the optimal solution can be achieved if P_b is printed first, as shown in Figure 6(b), where only two groups are required.

Considering the large optimization room, we mathematically model the group assignment problem. The problem is solved in two sub-steps to balance the solution quality and solving runtime. The first step is to minimize the total group number to pack as many patterns as possible into the same groups. By fixing the group number to the minimum value, which has drastically reduced the problem space, the second sub-step is to reassign the patterns to minimize the total required drying time. We implement a C++ program to build the corresponding integer linear programming (ILP) models for the two steps, and perform the optimization applying the state-of-the-art ILP solver Gurobi [1].

1.2.1 Group Number Minimization

For each pattern P_i, we maintain an integer variable g_i representing the group that the pattern is assigned to. The lower bound of g_i is set to 1 for the first group, and the upper bound of g_i is set to the number of patterns, which is the maximum possible number of groups when a group contains only one pattern. An integer variable g_{tot} is introduced to denote the total number of groups.

Modeling of Laplace Pressure Conflict: For each pair of patterns (P_i and P_j) that have the Laplace pressure conflict, the thin pattern (P_i) needs to be printed earlier than the thick pattern (P_j) to prevent possible ink absorption. The constraint can be simply modeled as:

$$g_i + 1 \leq g_j. \tag{1}$$

Modeling of Merging Conflict: For each pair of patterns (P_i and P_j) that have the merging conflict, they need to be assigned to different groups, implying that

$$g_i \neq g_j. \tag{2}$$

Since (2) is not a linear representation, we first transform it into:

$$g_i + 1 \leq g_j \ \vee \ g_i \geq g_j + 1, \tag{3}$$

which means that either P_i is printed earlier than P_j, or P_j is printed earlier than P_i. By applying an extremely large constant $\mathcal{M}$, i.e., the "Big M method" [2], we linearize (3) as:

$$g_i + 1 \leq g_j + \mathcal{M} \cdot q, \tag{4}$$

$$g_j + 1 \leq g_i + \mathcal{M} \cdot (1 - q), \tag{5}$$

where $\mathcal{M}$ is set to 1 million in our program, and q is a binary auxiliary variable. When q is set to 1, (4) becomes a tautology regardless of the value selection of g_i and g_j since they are always much smaller than $\mathcal{M}$. In the meanwhile, as the term $\mathcal{M} \cdot (1-q)$ in (5) becomes $\mathcal{M} \cdot 0 = 0$ and is thus removed, (5) becomes $g_j + 1 \leq g_i$, which indicates that P_j is printed earlier than P_i. By contrast, when q is set to 0, (4) becomes $g_i + 1 \leq g_j$ and (5) becomes a tautology so that P_i is printed earlier than P_j.

Optimization Objective: To confine the group number, for each pattern P_i, we build the following constraint

$$g_i \leq g_{tot}. \tag{6}$$

And the optimization problem can be modeled as:

Minimize: g_{tot}

Subject to: (1) and (4)–(6).

1.2.2 Drying Time Minimization

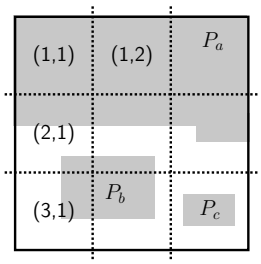
Fundamentally, the evaporation rate at a given point in the layout immediately after printing is governed by the local solvent concentration in the gas phase. This local solvent concentration, in turn, will depend on the mass transport of solvent in the system as well as the distribution of ink in the pattern. Consequently, given the minimized number of groups, there can be multiple conflict-free solutions that represent different drying times depending on the number, area, and position of objects printed in a given group.

In this sub-step, we search for a group assignment solution with the minimum required drying time based on the group number minimization result, i.e., with g_{tot} fixed to its minimum value. Since an under-drying point can cause the solvent concentration surrounding this point to increase, we model the local solvent concentration at a point as the summation of the solvent concentration increases caused by all under-drying points.

Since the longest drying time of one under-drying point dominates the required drying time of the pattern that the point belongs to, and the longest drying time of one pattern dominates the required drying time of the group that the pattern assigned to, minimizing the required drying time of a group in this sub-step is modeled as minimizing the longest drying time of any under-drying point belongs to this group.

The implementation of this sub-step consists of **area tiling**, **modeling of drying time increase**, and **mathematical model construction**, each of which will be explained in the following.

Area Tiling: To physically represent a point, we decompose the entire design into $\#tile_x \times \#tile_y$ square tiles. A smaller tile makes the model more accurate, but using smaller tiles implies a larger number of tiles to be modeled. Thus, this is

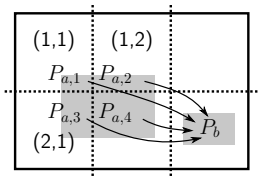


P_a: (1,1,1.00), (1,2,1.00), (1,3,1.00), (2,1,0.40), (2,2,0.40), (2,3,0.53)

P_b: (2,1,0.08), (2,2,0.16), (3,1,0.24), (3,2,0.48)

P_c: (3,3,0.27)

Figure 7: Illustration of tile occupancy.



P_a: (1,1,0.08), (1,2,0.16), (2,1,0.24), (2,2,0.48)

$dt_{a,2,3} = f_g(\sqrt{(2-1)^2+(3-1)^2}, 0.08) + f_g(\sqrt{(2-1)^2+(3-2)^2}, 0.16) + f_g(\sqrt{(2-2)^2+(3-1)^2}, 0.24) + f_g(\sqrt{(2-2)^2+(3-2)^2}, 0.48)$

Figure 8: The drying time increase ($dt_{a,2,3}$) of tile (2,3) caused by printing P_a. f_g is the Gaussian function that models the drying time increase and takes the tile distance as well as the occupancy ratio as the arguments.

a tradeoff between accuracy and the required computational effort. In our experiments, we set the total tile number to the scale of 10^4. The tile sidelength will then be calculated according to the design size.

Since a pattern may only occupy part of a tile, for each tile and each pattern covering that tile, we calculate a occupancy ratio. Figure 7 shows an example containing 9 tiles covered by 3 patterns. For each pattern, we maintain a set of tuples in a form as (m,n,λ), where (m,n) refers to the tile index and λ is the occupancy ratio.

Modeling of Drying Time Increase: To model the solvent concentration increase, for each tile containing part of a under-drying pattern, we build a time-invariant Gaussian function taking the distance from any tile to this tile as the first argument and the occupancy ratio as the second argument. The total required drying time of a tile is calculated by summing up the drying time increase caused by all tiles.

For implementation, we use the normal distribution function provided by the C++ standard library to construct the Gaussian function with the mean and the standard deviation set to 0 and $(\#tile_x + \#tile_y)/8$, respectively. A 3-D matrix is then constructed to denote the drying time increase for each tile caused by printing each pattern. We denote each entry as $dt_{i,m,n}$, where i is the pattern index (P_i) and (m,n) is the tile index. Figure 8 shows an example of calculating the drying time increase caused by printing P_a to the under-drying P_b. P_a is first decomposed into four parts with regard to the occupied tiles and the drying time increase of P_b caused by each part of P_a is summed up by considering the tile distance and the occupancy ratio.

Mathematical Model Construction: For each pattern and for each group, we introduce a binary variable $bg_{i,k}$ to denote whether P_i is assigned to the k-th group. The relation between $bg_{i,k}$ and g_i introduced in Section 1.2.1 can then be modeled as:

$$g_i = \sum_k k \times bg_{i,k}, \tag{7}$$

and for each pattern we need

$$\sum_k bg_{i,k} = 1, \tag{8}$$

to make exact one $bg_{i,k}$ be 1, i.e., each pattern is exactly assigned to one group.

Suppose the total required drying time of each tile for each group is denoted by $t_{k,m,n}$, where k is the group index and (m,n) is the tile index. $t_{k,m,n}$ can then be calculated by adding up the drying time increases caused by printing all the patterns assigned to this group:

$$t_{k,m,n} = \sum_i dt_{i,m,n} \times bg_{i,k}. \tag{9}$$

Since the total required drying time for each group is dominated by the maximum drying time among tiles, for each tile we introduce the following constraint:

$$t_{k,m,n} \le t_{k,max}, \tag{10}$$

where $t_{k,max}$ is an auxiliary variable to denote the maximum drying time for group k among all the tiles. In this way, to minimize the total required drying time to print all the patterns, we minimize the summation of the maximum drying time among the tiles for each group. The complete optimization problem can be modeled as:

$$\text{Minimize: } \sum_k t_{k,max}$$

Subject to: (1) and (4)–(10).

2 References

[1] Gurobi Optimization, Inc., *Gurobi Optimizer Reference Manual.* http://www.gurobi.com.

[2] S. P. Bradley, A. C. Hax, and T. L. Magnanti, *Applied Mathematical Programming.* Addison-Wesley Publishing Company, 1977.