Journal of Mathematical Imaging and Vision, 2, 39–50 (1992).

Local Decomposition of Gray-Scale Morphological Templates*

SAMER TAKRITI
University of Michigan, Ann Arbor, MI 48109-2117

PAUL D. GADER
Environmental Research Institute of Michigan, Ann Arbor, MI 48107-8618

Abstract. Template decomposition techniques can be useful for improving the efficiency of image-processing algorithms. The improved efficiency can be realized either by reorganizing a computation to fit a specialized structure, such as an image-processing pipeline, or by reducing the number of operations used. In this paper two techniques are described for decomposing templates into sequences of 3×3 templates with respect to gray-scale morphological operations. Both techniques use linear programming and are guaranteed to find a decomposition if one exists.

Key Words: image processing, linear programming, mathematical morphology, image algebra, template decomposition

1 Introduction

The increasing use of image-processing activities in the military, in industry, and in academia has led to the development of image algebra. Images are defined in terms of value sets and coordinate sets. A value set F is simply a set of values that an image could assume: real numbers, complex numbers, binary numbers of fixed length k, and extended real numbers. Coordinate sets are subsets of n-dimensional Euclidean space R^n. It follows from the definition that coordinate sets can be rectangular, hexagonal, and toroidal discrete arrays, as well as infinite subsets of R^n. Given coordinate and value sets X and F, respectively, an F-valued image a on X is the graph of a function $a : X \to F$. Thus, an F-valued image a on X is of the form $a = \{(x, a(x)) : a(x) \in F, \forall x \in X\}$. A generalized F-valued template t from Y to X is a function $t : Y \to F^X$. Thus for each $y \in Y$, $t(y) \in F^X$ or $t_y \equiv t(y) = \{(x, t_y(x)) : x \in X\}$. The point y is called the target point of the template t, and the values $t_y(x)$ are called the weights of the template t. More detailed definitions of templates and template operations are given in [1]–[3].

Several researchers and practitioners have studied the problem of template decomposition: given a template t, find a sequence of smaller templates $t_1, t_2, \ldots, t_n$ such that applying t to an image is equivalent to applying $t_1, t_2, \ldots, t_n$ sequentially to that image. In other words, t can be algebraically expressed in terms of $t_1, t_2, \ldots, t_n$. If interested, the reader can refer to [1], [2], and [4]–[13] to form a better understanding about the methods used.

In this paper two techniques are described for decomposing templates into sequences of 3×3 templates with respect to gray-scale morphological operations. We discuss dilation only. Decompositions with respect to erosion can be obtained from our discussion. Only translation-invariant templates will be considered. We shall restrict our attention to real-valued templates defined on one- and two-dimensional grids with rectangular supports.

* This work was supported in part by the U.S. Air Force Armament Laboratory of Eglin Air Force Base under contract F08635-89-C-0134.

2 Problem Description

Let x be an $m_x \times n_x$ template with weights $x_{i_x j_x}$, and let y be an $m_y \times n_y$ template with weights $y_{i_y j_y}$. The dilation of x and y is a template $t = x \boxtimes y$ of size $m_t \times n_t$, where $m_t = m_x + m_y - 1$ and $n_t = n_x + n_y - 1$. The relationship between x, y, and the weights $t_{i_t j_t}$ of t is given by

$$t_{i_t j_t} = \vee\{x_{i_x j_x} + y_{i_y j_y}\}$$
$$\text{where } i_y - i_x = i_t - \lceil m_t/2 \rceil$$
$$\text{and } j_y - j_x = j_t - \lceil n_t/2 \rceil,$$

where $\vee$ is the maximum in this case and $\lceil\ \rceil$ denotes the ceiling function. A more general and concise definition of the operation $\boxtimes$ is given in [3].

As described above, a gray-level dilation is a convolution operation with the sum of products replaced by the maximum of the sums. For example, if x and y are given by

$$x = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 2 & 1 & 5 \\ \hline 3 & 5 & 1 \\ \hline \end{array}, \quad y = \begin{array}{|c|c|c|}\hline 2 & 1 & 2 \\ \hline 1 & 2 & 3 \\ \hline 1 & 0 & 2 \\ \hline \end{array},$$

then $t = x \boxtimes y$ is given by

$$t = \begin{array}{|c|c|c|c|c|}\hline 3 & 7 & 6 & 7 & 5 \\ \hline 7 & 6 & 7 & 8 & 6 \\ \hline 6 & 7 & 8 & 8 & 5 \\ \hline 6 & 5 & 7 & 5 & 4 \\ \hline 4 & 3 & 5 & 4 & 3 \\ \hline \end{array}.$$

Consider the decomposition problem shown below:

$$\begin{array}{|c|c|c|c|c|}\hline t_{11} & t_{12} & t_{13} & t_{14} & t_{15} \\ \hline t_{21} & t_{22} & t_{23} & t_{24} & t_{25} \\ \hline t_{31} & t_{32} & t_{33} & t_{34} & t_{35} \\ \hline t_{41} & t_{42} & t_{43} & t_{44} & t_{45} \\ \hline t_{51} & t_{52} & t_{53} & t_{54} & t_{55} \\ \hline \end{array} =$$

$$\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array} \boxtimes \begin{array}{|c|c|c|}\hline y_{11} & y_{12} & y_{13} \\ \hline y_{21} & y_{22} & y_{23} \\ \hline y_{31} & y_{32} & y_{33} \\ \hline \end{array}.$$

The template t is known, and we wish to determine templates x and y that satisfy $t = x \boxtimes y$. The template equation leads to a set of linear constraints. To see this, consider the computation required to compute t_{21}:

$$t_{21} = \vee\{x_{33} + y_{21},\ x_{23} + y_{11}\}.$$

This equation can be written in the following form:

$$t_{21} \geq x_{33} + y_{21},$$
$$t_{21} \geq x_{23} + y_{11}.$$

Furthermore, one of the two inequality constraints must hold with equality. There is a set of linear inequalities of this form for every weight $t_{i_t j_t}$ of t. Call this set $\Omega_{i_t j_t}$. In general, decomposing the template $t_{m_t \times n_t}$ into $x_{m_x \times n_x}$ and $y_{m_y \times n_y}$ is equivalent to the mathematical problem of finding a solution for the system of inequalities

$$x_{i_x j_x} + y_{i_y j_y} \leq t_{i_t j_t}, \quad i_t = 1, 2, \ldots, m_t, \quad j_t = 1, 2, \ldots, n_t,$$

such that each group of inequalities $\Omega_{i_t j_t}$ has at least one constraint satisfied as an equality. We refer to the latter restriction as the *group equality restriction*.

We use a one-dimensional example to help describe the algorithms. If

$$t = \boxed{t_1}\boxed{t_2}\boxed{t_3}\boxed{t_4}\boxed{t_5},$$

then the problem is to find x and y such that $t = x \boxtimes y$. The system of inequalities corresponding to this problem is

$$\begin{array}{rrrrrrrrrrl}
 & x_3 & + & y_1 & & & & & & \leq & t_1, \quad \text{(I.1.1)} \\
\hline
x_2 & & + & y_1 & & & & & & \leq & t_2, \quad \text{(I.2.1)} \\
 & & & & & & & & & & \\
 & x_3 & & & + & y_2 & & & & \leq & t_2, \quad \text{(I.2.2)} \\
\hline
x_1 & & + & y_1 & & & & & & \leq & t_3, \quad \text{(I.3.1)} \\
 & & & & & & & & & & \\
 x_2 & & & & + & y_2 & & & & \leq & t_3, \quad \text{(I.3.2)} \\
 & & & & & & & & & & \\
 & x_3 & & & & & + & y_3 & & \leq & t_3, \quad \text{(I.3.3)} \\
\hline
x_1 & & & & + & y_2 & & & & \leq & t_4, \quad \text{(I.4.1)} \\
 & & & & & & & & & & \\
 x_2 & & & & & & + & y_3 & & \leq & t_4, \quad \text{(I.4.2)} \\
\hline
x_1 & & & & & & + & y_3 & & \leq & t_5. \quad \text{(I.5.1)}
\end{array}$$

The group equality restriction can be written in the form

$$\begin{array}{rrrrrrrrrrl}
 & x_3 & + & y_1 & & & & & & = & t_1, \quad \text{(II.1.1)} \\
\hline
 & x_2 & & + & y_1 & & & & & = & t_2, \quad \text{(II.2.1)} \\
\text{or} & & & & & & & & & & \\
 & x_3 & & & & + & y_2 & & & = & t_2, \quad \text{(II.2.2)} \\
\hline
 x_1 & & & + & y_1 & & & & & = & t_3, \quad \text{(II.3.1)} \\
\text{or} & & & & & & & & & & \\
 & x_2 & & & & + & y_2 & & & = & t_3, \quad \text{(II.3.2)} \\
\text{or} & & & & & & & & & & \\
 & x_3 & & & & & & + & y_3 & = & t_3, \quad \text{(II.3.3)} \\
\hline
 x_1 & & & & & + & y_2 & & & = & t_4, \quad \text{(II.4.1)} \\
\text{or} & & & & & & & & & & \\
 & x_2 & & & & & & + & y_3 & = & t_4, \quad \text{(II.4.2)} \\
\hline
 x_1 & & & & & & & + & y_3 & = & t_5. \quad \text{(II.5.1)}
\end{array}$$

We describe two algorithms for solving the problem. The first algorithm uses a depth-first search technique that is efficient in terms of storage. The second algorithm uses an integer-programming approach that, in turn, is solved by using a branch-and-bound technique.

3 Depth-First Search Decomposition Algorithm

Let π be the problem of finding a feasible solution for the system of linear inequalities (I) that also satisfies (II). An algorithm for solving π based on a depth-first search in conjunction with linear programming has been developed. Some relevant linear-programming principles and properties of the set of feasible solutions of π are described in appendix A. See [14] for more detailed explanation on this topic.

Each node in the search tree represents the feasible region corresponding to a set of linear constraints. The root of the tree represents the set of feasible solutions for (I). Successor nodes in the tree are obtained by adding a suitable constraint from (II) to the set of constraints used in the immediate predecessor. At each node linear programming is used to test for the existence of a feasible solution for the set of linear constraints. The performance of the algorithm is independent of the particular feasible solution selected from the set of feasible solutions at that node. The algorithm is based on the following remarks:

1. When one is looking for a feasible solution for π, it is sufficient to consider the cases for which only one of the constraints in each group of (I) is an equality.
2. If at a certain level of search, say, level n, there is no feasible solution for the problem π_n, then none of the descendants of π_n is feasible. Therefore there is no need to search among these descendants. The proof is simple: the set of feasible solutions of any descendant of π_n is a subset of the set of feasible solutions of π_n itself. If any of the

descendants is feasible, then π_n must also be feasible.

We describe the algorithm by using the one-dimensional problem as an example, i.e., $m_x = m_y = m_t = 1$. The algorithm begins by finding a feasible solution for (I). This problem, call it π_1, is always feasible. The problem π_2 is then formed by adding the constraint (II.1.1) to the set of constraints of π_1. Note that the first group of inequalities, i.e., (I.1), has only one constraint. It follows that if π_2 is infeasible, then π is also infeasible. On the other hand, if π_2 is feasible and the current solution is not feasible for π, more equality constraints from (II) must be added. This is done by adding a constraint from (II.2) to the set of constraints of π_2. Let this constraint be (II.2.1), and call this problem π_3. If π_3 is feasible, the algorithm proceeds down the tree as described above. If, on the other hand, π_3 has no feasible solution, the algorithm creates problem π_4, which is obtained by adding constraint (II.2.2) to the set of constraints of problem π_2. If π_4 is infeasible, then π is infeasible. Otherwise, the algorithm proceeds to the next level. The search terminates when the algorithm finds a feasible solution for π if there is one.

Thus the rule at any node can be summarized by the *node-level search rule:*

If the current problem is feasible,
  then set current problem to current problem plus a previously unselected constraint from the next group and descend to the next level;
  else
    if there are previously unselected constraints in the same group,
      then set current problem to current problem plus one such constraint and test for feasibility;
      else set current node to the direct predecessor of current node.

The method of selection of equality constraints from (II) is based on order of appearance. For example, if the last selected constraint at a certain node is (II.$i.j$), then the next selected constraint is (II.$i.j+1$) if $j+1$ is less than or equal to the number of constraints in group i. This method of selection has been arbitrarily chosen, and more research must be done to find better selection methods. The following is a detailed description of the algorithm:

- *Initialization.* $n = 0$, $k_i = 1$, $i = 1, 2, \ldots,$ number of groups. Find a feasible solution for (I). Let the set of linear constraints in the current node Γ =(I).
- *General step.*
  1. $n = n + 1$.
  2. Let i, j be the indices of x, y corresponding to constraint k_n in group n. Find a feasible solution for Γ under the additional constraint $x_i + y_j = t_n$. If a feasible solution exists, go to step 6.
  3. Change the constraint k_n in group n of Γ to an inequality constraint.
  4. $k_n = k_n + 1$. If k_n is less than or equal to the number of constraints in group n, go to step 2.
  5. Set $k_n = 1$, $n = n - 1$. If $n = 0$, stop; the given problem has no feasible solution. Otherwise, go to step 4.
  6. Compute

$$\delta = \sum_{l=1}^{n_t} \min_{ij} \{t_l - x_i - y_j \mid x_i + y_j \leq t_l \text{ is a constraint in } \Omega_l\}.$$

     If $\delta = 0$, then stop; the current solution is a feasible solution for the given problem. Otherwise, go to step 7.
  7. Change constraint k_n in group n of Γ to an equality constraint. Go to step 1.

This algorithm is guaranteed to find a solution to the template-decomposition problem if a solution exists. In the worst case the algorithm performs an exhaustive search of the finite set of all possible combinations of equality constraints. This may occur when π has no feasible solution. However, the algorithm can determine the nonexistence of a solution quite

rapidly without exhausting all possible combinations. A breadth-first search algorithm was tried and was found to be inferior to the depth-first search algorithm.

Linear programming is used to check the feasibility in step 2. The linear-programming algorithm used, call it PDAT, is described in appendix A. It has a major advantage over other known linear-programming algorithms. Assume that the current node π_n is feasible, and let π_{n+1} be a direct successor of π_n. PDAT looks for a feasible solution for π_{n+1} in the set of feasible solutions for π_n. This implies that the solution produced by PDAT satisfies the constraints of π_n whether π_{n+1} is feasible or not. In other words, the current solution is feasible for all ancestors of the current node. In this case, only the current solution has to be stored. This makes the depth-first search algorithm applicable on microcomputers.

4 Numerical Example

Let $t = [8, 10, 9, 9, 2]$. It is desired to decompose t into two templates x and y each of dimension 1×3.

0. $n = 0$, $k_i = 1$, $i = 1, 2, \ldots, 5$. $x = [2, 9, 8]$, $y = [0, 0, 0]$ is a feasible solution for (I).
1. $n = 1$.
2. Find a feasible solution for (I) with the additional constraint $x_3 + y_1 = 8$. The solution is $x = [2, 9, 8]$, $y = [0, 0, 0]$.
6. $\delta = 1$.
7. Change constraint 1 in group 1 into an equality constraint.
1. $n = 2$.
2. Find a feasible solution for (I) with the additional constraint $x_2 + y_1 = 10$. The solution is $x = [2, 10, 8]$, $y = [0, -1, -1]$.
6. $\delta = 1$.
7. Change constraint 1 in group 2 into an equality constraint.
1. $n = 3$.
2. Find a feasible solution for (I) with the additional constraint $x_1 + y_1 = 9$. The solution is $x = [9, 10, 8]$, $y = [0, -1, -7]$.
6. $\delta = 1$.
7. Change constraint 1 in group 3 into an equality constraint.
1. $n = 4$.
2. Find a feasible solution for (I) with the additional constraint $x_1 + y_2 = 9$. No feasible solution exists.
3. $k_4 = 2$.
2. Find a feasible solution for (I) with the additional constraint $x_2 + y_3 = 9$. There is no feasible solution.
3. $k_4 = 3$.
4. $k_4 = 1$, $n = 3$.
5. Change the constraint 1 in group 3 into an inequality constraint.
3. $k_3 = 2$.
2. Find a feasible solution for (I) with the additional constraint $x_2 + y_2 = 9$. The solution is $x = [9, 10, 8]$, $y = [0, -1, -7]$.
6. $\delta = 1$.
7. Change constraint 2 in group 3 into an equality constraint.
1. $n = 4$.
2. Find a feasible solution for (I) with the additional constraint $x_1 + y_2 = 9$. There is no feasible solution.
3. $k_4 = 2$.
2. Find a feasible solution for (I) with the additional constraint $x_2 + y_3 = 9$. The solution is $x = [9, 16, 14]$, $y = [-6, -7, -7]$.
6. $\delta = 0$. Stop. The current solution is a feasible solution for the given problem.

Figure 1 illustrates the search tree of the previous example.

5 Numerical Results

The algorithm has been implemented by using the Pascal language. A problem generator was coded to create some test problems for the algorithm. This generator fills each cell in the x template and the y template with a random inte-

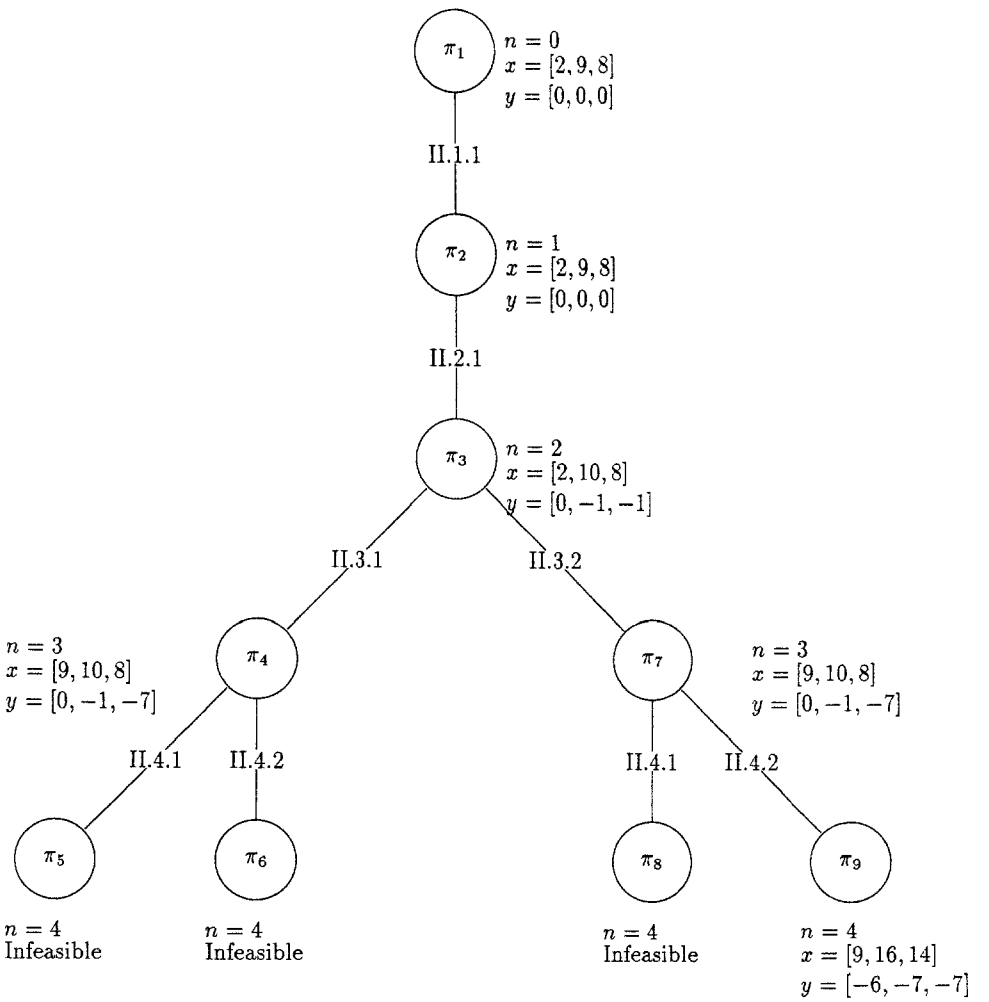


Fig. 1. Tree followed by the depth-first search algorithm.

ger number. The x and y are then composed to obtain the corresponding t template. So that we could get a better idea about the performance of the algorithm, 100 problems were randomly generated and fully solved. The size of the t template in each problem was 9×9, and the task was to decompose t into two templates, each of dimension 5×5. The number of nodes studied and the solution times varied over a large range, depending on the elements in template t. The results are shown in table 1.

Table 1. Number of nodes studied by the depth-first search algorithm to decompose a 9×9 template into two 5×5 templates.

Number of Nodes	Frequency
100–1000	50
1000–10 000	27
10 000–100 000	15
≥100 000	8
Total	100

The previous problems were solved on an Apple Macintosh SE/30 computer. The time required to evaluate each node, i.e., to check whether the set of constraints at that node is feasible or not and to decide about the set of constraints to be used in the next node, was 38 ms.

6 Integer-Programming Formulation

The problem of decomposing templates into a product of two other templates can be formulated as an integer-programming problem. The integer program is guaranteed to find a solution if there is one. Here also, we use the one-dimensional problem as an example.

The problem π can be written in the following form:

$$
\begin{array}{l}
x_3 + y_1 \le t_1, \\
x_3 + y_1 + M z_{11} \ge t_1, \\
\hline
x_2 + y_1 \le t_2, \\
x_2 + y_1 + M z_{21} \ge t_2, \\
x_3 + y_2 \le t_2, \\
x_3 + y_2 + M z_{22} \ge t_2, \\
\hline
x_1 + y_1 \le t_3, \\
x_1 + y_1 + M z_{31} \ge t_3, \\
x_2 + y_2 \le t_3, \\
x_2 + y_2 + M z_{32} \ge t_3, \\
x_3 + y_3 \le t_3, \\
x_3 + y_3 + M z_{33} \ge t_3, \\
\hline
x_1 + y_2 \le t_4, \\
x_1 + y_2 + M z_{41} \ge t_4, \\
x_2 + y_3 \le t_4, \\
x_2 + y_3 + M z_{42} \ge t_4, \\
\hline
x_1 + y_3 \le t_5, \\
x_1 + y_3 + M z_{51} \ge t_5,
\end{array}
$$

$$
\begin{aligned}
z_{11} &\le 0, \\
z_{21} + z_{22} &\le 1, \\
z_{31} + z_{32} + z_{33} &\le 2, \\
z_{41} + z_{42} &\le 1, \\
z_{51} &\le 0, \\
z_{ij} &= 0 \text{ or } 1,
\end{aligned}
$$

where M is a large number and z_{ij} is a 0/1 or yes/no variable. When z_{ij} is equal to 0, the constraint number j in group i must be satisfied as an equality.

To have at least one equality constraint in each group Ω_i, the constraint

$$\sum_j z_{ij} \le (\text{number of constraints in group } i) - 1$$

is added. M must be chosen in a way such that if z is equal to 1 in one of the constraints,

then the corresponding inequality $x + y + M \geq t$ must be satisfied no matter what the values of x, y, and t are. From the previous inequality the value of M must be greater than $\max\{t\} - \min\{x\} - \min\{y\}$. By giving M the value of $\max\{t\} - \min\{x\} - \min\{y\} + 1$, it is guaranteed to satisfy the aforementioned constraint. The problem here is to estimate the values of $\min\{x\}$ and $\min\{y\}$. There is no clear method to perform the previous estimation, but finding lower bounds on the values of x and y does not appear to be a very difficult task.

In the previous integer program there is no objective function to be optimized. The goal is to find a feasible solution for the given system that, in turn, corresponds to a feasible solution for π. Any integer-programming method can be used to solve the resulting program, and the branch-and-bound method has been used in this case. In general, this formulation has been found to be very efficient with smaller problems. The disadvantage of the integer-programming formulation is the large amount of storage required to solve the problem on a digital computer because of the large number of nodes in the search tree that must be stored.

7 Generalization: Decomposition of Arbitrarily Sized Templates into 3×3 Templates

Both the depth-first search algorithm and the integer-programming algorithm can be applied, without change, to decompose a template $t_{m_t \geq 3, n_t \geq 3}$ into a sequence of 3×3 templates. The following pseudocode performs such a decomposition.

- *Initialization.* $i = 1$, $k = 1$. Let the current template be $t^i_{m_{t^i} n_{t^i}} = t_{m_t n_t}$.
- *General step.*
  1. If $m_{t^i} = n_{t^i} = 3$, stop; the sequence of templates t^j, $j = i, i+1, \ldots, k$ is a decomposition for $t_{m_t n_t}$. Otherwise, go to step 2.
  2. Decompose $t^i_{m_{t^i}, n_{t^i}}$ into two templates x^i and y^i each of size $\max\{3, \lceil m_{t^i}/2 \rceil\} \times \max\{3, \lceil n_{t^i}/2 \rceil\}$ as follows:
     Write the set of linear constraints corresponding to the problem $t^i_{m_{t^i}, n_{t^i}} = x^i_{m_{x^i}, n_{x^i}} \boxtimes y^i_{m_{y^i}, n_{y^i}}$, using the relationship described at the beginning of section 2. Solve this problem by using one of the previous algorithms. If no feasible solution exists, stop; the given problem is infeasible. Otherwise, to go step 3.
  3. $t^{k+1} = x^i$, $t^{k+2} = y^i$, $k = k + 2$, $i = i + 1$. Go to 1.

8 Conclusions

Two algorithms have been developed to solve the template-decomposition problem. They are guaranteed to find a solution for the problem if one exists. Both algorithms use linear programming to evaluate the nodes of the search trees. Future research directions include developing relationships between the geometry of the linear-programming problem and the shape of the templates, improving the search-tree pruning, and generalizing the approach to templates with nonrectangular support. This last area would involve merging decomposition of binary morphological templates with the gray-scale decompositions.

Appendix A: Linear-Programming Concepts

A linear-programming problem is an optimization problem. The objective is to maximize or minimize a linear function subject to a set of linear constraints. A linear program can be written in the mathematical form $\max_x\{c^T x | Ax \leq b\}$, where $c \in R^n$ is the objective function, $x \in R^n$ is the vector of decision variables, $A \in R^{m \times n}$ is a given matrix, and $b \in R^m$ is the right-hand-side vector. A vector x that satisfies all the constraints of a linear program is called a feasible solution for the linear program, and the set of all feasible solutions is called the feasible region. At each node of the search tree of section 3, we must check for the existence of a feasible solution for the set of linear constraints at that node.

Associated with any linear program there is another linear program called the dual. For

example, if the given linear program, the primal, is $\max_x\{c^T x | Ax \leq b\}$, then the corresponding dual is $\min_\pi\{\pi b | \pi A = c, \pi \geq 0\}$, where $\pi \in R^m$ is a row vector. π_i is called the dual variable associated with the primal constraint i. The fundamental duality theorem states that if either the primal or the dual problem has an optimal feasible solution, then the other does also and the two optimal objective values are equal. As a result to the previous property, solving the primal problem is equivalent to solving the dual problem, and this result can be applied to our problem. By rearranging the rows of inequality (I), it can be written in the following form:

$$
\begin{aligned}
x_1 \quad &+ y_1 \quad &&\leq t_3 \text{ (I.3.1) } u_{11},\\
x_1 \quad &\quad + y_2 \quad &&\leq t_4 \text{ (I.4.1) } u_{12},\\
x_1 \quad &\quad \quad + y_3 &&\leq t_5 \text{ (I.5.1) } u_{13},\\
x_2 \quad &+ y_1 \quad &&\leq t_2 \text{ (I.2.1) } u_{21},\\
x_2 \quad &\quad + y_2 \quad &&\leq t_3 \text{ (I.3.2) } u_{22},\\
x_2 \quad &\quad \quad + y_3 &&\leq t_4 \text{ (I.4.2) } u_{23},\\
x_3 \quad &+ y_1 \quad &&\leq t_1 \text{ (I.1.1) } u_{31},\\
x_3 \quad &\quad + y_2 \quad &&\leq t_2 \text{ (I.2.2) } u_{32},\\
x_3 \quad &\quad \quad + y_3 &&\leq t_3 \text{ (I.3.3) } u_{33},
\end{aligned}
$$

where u_{ij} are the dual variables corresponding to the constraints of the original system.

Looking for a feasible solution for the previous system subject to an additional equality constraint is equivalent to maximizing the sum of the variables in the left-hand side of the equality constraint subject to inequality (I). For example, if the additional constraint is $x_1 + y_1 = t_3$, then the existence of a feasible solution can be checked for by maximizing the function $x_1 + y_1$ subject to the previous set of linear constraints. If, after the problem is solved, the optimal objective function value equals to t_3, then the given problem has a feasible solution. On the other hand, if $x_1 + y_1 < t_3$, then the given problem has no feasible solution. Writing the dual of the previous problem leads to the following linear program:

Minimize

$$t_3u_{11} + t_4u_{12} + t_5u_{13} + t_2u_{21} + t_3u_{22} + t_4u_{23} + t_1u_{31} + t_2u_{32} + t_3u_{33}$$

subject to

$$
\begin{aligned}
u_{11} + u_{12} + u_{13} &= 1,\\
u_{21} + u_{22} + u_{23} &= 0,\\
u_{31} + u_{32} + u_{33} &= 0,\\
u_{11} + u_{21} + u_{31} &= 1,\\
u_{12} + u_{22} + u_{32} &= 0,\\
u_{13} + u_{23} + u_{33} &= 0,\\
u_{ij} &\geq 0.
\end{aligned}
$$

It is obvious that the dual has a very special structure. Problems with such a structure are called transportation problems. The transportation problem can be stored in an efficient form by using the transportation matrix. In this matrix the variable u_{ij} corresponds to the cell (i, j). The constraints specify the sum of the variables in each column and row in the array. The transportation problem has the following integer property: If all the elements in the right-hand side of the equality constraints are positive integers, then at least one optimum solution of that problem is an integer vector. The following is a description of a primal–dual algorithm for solving the uncapacitated balanced transportation problem efficiently. The given problem is of the following form:

$$\text{Minimize } \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

subject to

$$
\begin{aligned}
\sum_{j=1}^{m} x_{ij} &= a_i, \quad i = 1, 2, \ldots, n,\\
\sum_{i=1}^{n} x_{ij} &= b_j, \quad j = 1, 2, \ldots, m,\\
x_{ij} &\geq 0 \quad \text{for all } i, j,
\end{aligned}
$$

where $a_i, b_j \geq 0$ for all i, j and $\sum_i a_i = \sum_j b_j$. The dual problem is the following:

$$\text{Maximize } \sum_i a_iu_i + \sum_j b_jv_j$$

subject to

$$u_i + v_j \leq c_{ij} \quad \text{for all } i, j.$$

The algorithm is as follows.

- *Initialization.* Find a dual feasible solution for the given problem. This can be done by selecting the dual solution to be $u_i = \min\{c_{ij} | j = 1, \ldots, n\}$, $v_j = \min\{c_{ij} - u_i | i = 1, \ldots, m\}$. Let $\overline{c_{ij}} = c_{ij} - u_i - v_j$, $x_{ij} = 0$ for all cells (i, j).
- *General step.*
  1. Identify all rows i satisfying $\sum_j x_{ij} < a_i$, and label all such rows with label $(s, +)$. All of these rows are now labeled and unscanned.
  2. If the list of labeled and unscanned rows and columns is empty at this stage, go to step 4. Otherwise, select a row or a column for scanning, and scan it.

     To scan row i, label each column j, where j is such that column j is unlabeled so far and $\overline{c_{ij}} = 0$ with the label (row i, $+$).

     To scan column j, label all rows i, where i is such that row i is unlabeled so far and $x_{ij} > 0$ with the label (column j, $-$).

     The rows or columns that are newly labeled in this step are at present labeled and unscanned. The row or column that is scanned in this step is now labeled and scanned. Go to step 3.
  3. Check whether any of the columns with unfulfilled requirements, i.e., a column j such that $\sum_i x_{ij} < b_j$, is labeled. If such a column is labeled, go to step 5. If none of the columns with unfulfilled requirements have been labeled, go to step 2.
  4. Let $\delta = \min\{\overline{c_{ij}} |$ row i is a labeled row, column j is an unlabeled column$\}$. Let the new dual feasible solution be (u, v) where

$$\begin{aligned} u_i &= u_i + \delta \quad \text{if row } i \text{ is labeled,} \\ &= u_i \quad\quad \text{if row } i \text{ is unlabeled,} \\ v_j &= v_j - \delta \quad \text{if column } j \text{ is labeled,} \\ &= v_j \quad\quad \text{if column } j \text{ is unlabeled.} \end{aligned}$$

     Keep all the labeled columns as labeled and scanned. Change the state of each labeled row into labeled and unscanned. Update $\overline{c_{ij}}$. Go to step 2.
  5. Let column j_1 be the column whose labeling led to this step. Suppose that the label on column j_1 is (row i_1, $+$). Add θ to the present amount of flow in cell (i_1, j_1). Now look at the label on row i_1. Suppose it is (column j_2, $-$). Add a $-\theta$ to the present flow in cell (i_1, j_2). Now look at the label on column j_2, and so on. Repeat, adding θ and $-\theta$ alternately to the flow amounts in the cells indicated by the labels until a row, say, row i_k, with a label of $(s, +)$ is reached. Let $\theta = \min\{a_i - \sum_k x_{ik}, b_j - \sum_k x_{kj}$ over all cells (i, j) with a $-\theta$ alteration$\}$. Substitute the value of θ to get the new x. If x fulfills the requirements in all the columns, i.e., $\sum_i x_{ij} = b_j$ for all columns j, it is an optimum feasible solution of the transportation problem and the algorithm is terminated. Otherwise, erase the old labels on all the rows and the columns. Go to step 1.

A complete study of this subject can be found in [15].

Appendix B: Properties of the Set of Feasible Solutions of π

It is worthwhile to note the following properties of the set of feasible solutions of π:

1. If t is a given template with integer weights and if the corresponding π has a feasible solution, then there exists an integer feasible solution for the problem π. This follows from the integer property of the transportation problem.
2. If (x_0, y_0) is a feasible solution for π, then $(x_0 + \delta, y_0 - \delta)$ is also a feasible solution, where δ is any real number. This implies that either π has no feasible solution at all or that it has an infinite number of solutions.
3. The set of feasible solutions for π is not necessarily convex. For instance, let $t = [1, 2, 1, 2, 1]$. We want to decompose this template into two templates x and y each

of dimension 1×3. A feasible solution for this problem would be $x_1 = [1, 0, 1]$, $y_1 = [0, 1, 0]$. Also, $x_2 = [1, 2, 1]$, $y_2 = [0, -1, 0]$ is another feasible solution. Let us take the point (x, y) that is in the middle of the line segment joining (x_1, y_1) and (x_2, y_2). Obviously, $x = [1, 1, 1]$, $y = [0, 0, 0]$ is not a feasible solution of the given problem.

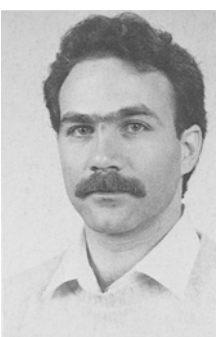
Acknowledgments

We would like to thank Patrick Coffield, Karen Norris, and Sam Lambert for their support. We would also like to thank Professor Katta Murty of the University of Michigan for his help.

References

1. J.L. Davidson, "*Lattice Structures in the Image Algebra and Applications to Image Algebra*," Ph.D. dissertation, **University of Florida**, Gainesville, FL., 1989.
2. P.D. Gader, "*Image Algebra Techniques for Parallel Computation of Discrete Fourier Transforms and General Linear Transforms*," Ph.D. dissertation, **University of Florida**, Gainesville, FL., 1986.
3. G.X. Ritter, J.N. Wilson, and J.L. Davidson, "*Image Algebra: An overview,*" *Comput. Vis. Graph., Image Process.*, vol. 49, 1990, pp. 297–331.
4. J.L. Davidson, "*Local Decomposition of Invariant Lattice Transforms,*" in *Image Algebra and Morphological Image Processing*, P.D. Gader, ed., *Proc. Soc. Photo-Opt. Instrum. Engrg.*, vol. 1350, 1990, pp. 443–454.
5. P.D. Gader, "*Separable Decompositions and Approximations of Gray Scale Morphological Templates*," *Comput. Vis., Graph., Image Process.*, vol. 53, 1991, pp. 288–296.
6. P.D. Gader and E.G. Dunn, "*Image Algebra and Morphological Template Decomposition,*" in *Aerospace Pattern Recognition*, M.R. Weathersby, ed., *Proc. Soc. Photo-Opt. Instrum. Engrg.*, vol. 1098, 1989, pp. 134–146.
7. P.D. Gader and S. Takriti, "*Decomposition Techniques for Gray-Scale Morphological Templates*," in *Image Algebra and Morphological Image Processing*, P.D. Gader, ed., *Proc. Soc. Photo-Opt. Instrum. Engrg.*, vol. 1350, 1990, pp. 431–443.
8. R.M. Haralick and X. Zhuang, "*Morphological Structuring Element Decomposition,*" *Comput. Vis., Graph., Image Process.*, vol. 35, 1986, pp. 370–382.
9. T.M. Kanungo, R.M. Haralick, and X. Zhuang, "*B-Code Dilation and Structuring Element Decomposition for Restricted Convex Shapes*," in *Image Algebra and Morphological Image Processing*, P.D. Gader, ed., *Proc. Soc. Photo-Opt. Instrum. Engrg.*, vol. 1350, 1990, pp. 419–430.
10. D. Li and G.X. Ritter, "*Decomposition of Separable and Symmetric Convex Templates*," in *Image Algebra and Morphological Image Processing*, P.D. Gader, ed., *Proc. Soc. Photo-Opt. Instrum. Engg.*, vol. 1350, 1990, pp. 408–418.
11. O.R. Mitchell and F.Y. Shih, "*Decomposition of Gray-Scale Morphological Structuring Elements*," in *Proc. IEEE 1987 Workshop on Computer Vision*, 1987, pp. 304–306.
12. O.R. Mitchell and F.Y. Shih, "*Threshold Decomposition of Gray-Scale Morphology into Binary Morphology,*" *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. PAMI-11, 1989, pp. 31–42.
13. G.X. Ritter and P.D. Gader, "*Image Algebra Techniques for Parallel Image Processing,*" *J. Parallel Distr. Comput.*, vol. 4(5), 1987, pp. 7–44.
14. K.G. Murty, *Linear Programming*, New York: John Wiley, 1983.
15. K.G. Murty, *Network Programming*, New Jersey: Prentice Hall, 1992.

Samer Takriti is a Ph.D. student in the Department of Industrial and Operations Engineering at the University of Michigan. He received his B.Sc. degree in Civil Engineering from the University of Damascus, Syria, in 1986 and his M.Sc. degree in civil engineering from the University of Michigan in 1988. His current research involves the optimization of stochastic linear programs.



Paul Gader received his Ph.D. in applied mathematics from the University of Florida in 1986. Since then Dr. Gader has held positions as section manager and research engineer at the Environmental Research Institute of Michigan, se-

nior research scientist at Honeywell Systems and Research Center, and as assistant professor of mathematics at the University of Wisconsin, Oshkosh. He is currently an assistant professor in the Department of Computer and Electical Engineering at the University of Missouri, Columbia. He has performed research in applied mathematics, image algebra and mathematical morphology, obstacle detection, scene analysis, automatic target recognition, and recognition of handwritten ZIP codes and words.