Connectivity Preserving Voxel Transformation

Anvesh Komuravelli[1], Arnab Sinha[2], and Arijit Bishnu[1]

[1] Computer Science and Engineering Department, Indian Institute of Technology, Kharagpur, Kharagpur-721302, India
{anvesh,bishnu}@cse.iitkgp.ernet.in
[2] Dept. of Electrical Engineering, Princeton University, Princeton, New Jersey, USA-08544
sinha@princeton.edu

Abstract. A three dimensional digital binary image is $\mathbf{B}_{26}$ connected if its set of black voxels is 26-connected, i.e. for all black voxels there exists at least one black voxel among its 26 neighbors. We show that any two such images I and J of c_1 and c_2 number of connected components respectively and n voxels each, can be transformed into one another maintaining the $\mathbf{B}_{26}$ connectivity of the black voxels by $O((c_1 + c_2)n^2)$ interchanges.

1 Introduction

A three (two) dimensional digital binary image (I) is a function $I : \mathbb{Z}^3 \rightarrow \{0, 1\}$ ($I : \mathbb{Z}^2 \rightarrow \{0, 1\}$). Any element in $\mathbb{Z}^3$ ($\mathbb{Z}^2$) is called a *voxel* (*pixel*). We consider finitely many lattice points (voxels/pixels) from $\mathbb{Z}^3$ ($\mathbb{Z}^2$) in I. A voxel (pixel) p is black (white) if $I(p) = 1$ ($I(p) = 0$). We call two pixels (x_1, y_1) and (x_2, y_2) to be 8-neighbors if and only if $(x_1 - x_2)^2 + (y_1 - y_2)^2 \leq 2$. For 4-neighbors, it is $(x_1 - x_2)^2 + (y_1 - y_2)^2 \leq 1$. We call two voxels (x_1, y_1, z_1) and (x_2, y_2, z_2) to be 26-neighbors if and only if $(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \leq 3$. This induces a graph G_{26} whose vertex set is $\mathbb{Z}^3$ and there exist edges between two lattice points satisfying the above inequality. In a 3-D binary image I, $\mathbf{B}_{26}$ is a sub-graph of G_{26} induced by the black voxels in I. Similar graphs can be defined for 2-D binary images also. A pair of neighboring (4 or 8) [4] opposite-valued pixels in a 2-D binary image I is called *interchangeable* if reversing their values preserves the topology of the image [5,6]. The interchange does not affect the number of 0s and 1s in I. We will define interchangeable voxel pair later on. Two 2-D binary images I and J are called *IP-equivalent* [5,6] if there exists a sequence of binary images $I = I_0, I_1, \ldots, I_i, \ldots, I_k = J$ such that any I_i $(1 \leq i \leq k)$ can be obtained from I_{i-1} by reversing an *interchangeable* pixel pair. Rosenfeld and Nakamura [6] proved the conjecture made in [5] that if two binary images I and J have two simply connected sets S and T respectively of the same number of 1s, then I and J are *IP-equivalent*. In a recent comprehensive work that also deals with the combinatorial bounds on the number of interchanges, Bose et al. [1] generalized the results in [6]. They showed that for any $(a, b) \in \{(4, 8), (8, 4), (8, 8)\}$, any two $\mathbf{B}_a, \mathbf{W}_b$-connected images I and J each with n

V.E. Brimkov, R.P. Barneva, H.A. Hauptman (Eds.): IWCIA 2008, LNCS 4958, pp. 1–12, 2008.

black pixels differ by a sequence of $O(n^2)$ interchanges. The corresponding result for two $\mathbf{B}_4, \mathbf{W}_4$ connected images is $O(n^4)$. A binary image I is called $\mathbf{B}_a, \mathbf{W}_b$ $(a, b \in \{4, 8\})$ connected if its foreground (1) is a-connected and its background (0) is b-connected. The interchanges considered by Bose et al. [1] are also of *interchangeable* pixel pairs such that the connectivity of both foreground and background are maintained as in [5,6]. This sort of transformation problems has motivation in robotics [2] where researchers are interested in the number of moves needed in going from a configuration to another under some restrictions in the movement patterns. Under a more restricted and complex interchange rule, Dumitrescu and Pach [3] show that any two $\mathbf{B}_4$ connected images are apart by $O(n^2)$ interchanges where an interchange takes place between two 8-neighbor pixels such that the image obtained after the interchange is still $\mathbf{B}_4$ connected. Though Dumitrescu and Pach talk of modular metamorphic systems in terms of motion planning in [3], a connection to pixels is straightforward.

In this work, we consider connectivity preserving voxel transformation of 3-D binary images under a very relaxed and simple model of connectivity. To the best of our knowledge, connectivity preserving voxel transformation has not been considered earlier. Section 2 discusses preliminaries needed for our work. Section 3 discusses the main body of our work. In Section 4, we discuss connectivity preserving voxel transformation under the model proposed in [2] for 2-D.

2 Preliminaries

2.1 Definition and Notations

Our model is simple as we do not consider the connectivity of the white voxels. The earlier works in 2-D [5,6,1] consider connectivity of both black and white pixels. We think considering this simple model is worthwhile for an initial study on connectivity preserving voxel transformation.

We call a pair of neighboring (26) opposite-valued voxels in I *interchangeable* if reversing their values preserves the $\mathbf{B}_{26}$ connectivity of the 3-D image. The interchange obviously does not affect the number of 0s and 1s in I. Two $\mathbf{B}_{26}$ connected 3-D binary images I and J of the same number of voxels are called *transformable* if there exists a sequence of 3-D binary images $I = I_0, I_1, \ldots, I_i, \ldots, I_k = J$ such that any I_i $(1 \leq i \leq k)$ can be obtained from I_{i-1} by reversing an *interchangeable* voxel pair. We show in this work that any two 3-D binary images I and J which are $\mathbf{B}_{26}$ connected and have the same number of voxels are transformable. We do this by transforming I to a linear chain of voxels. So, it follows that J can also be transformed to a linear chain of voxels; and the transformation of I to J can be obtained by transforming I to a linear chain of voxels and then retracing the transformation (of J) from the linear chain of voxels back to J.

Following are the definitions of the terms we will be using throughout. See Fig. 1. When a voxel moves because of the interchanges such that its z-coordinate remains unaffected, we use the term *pixel* also. In the body of the text, we interchangeably use the term *voxel* and *pixel*.

Layer: A 3-D object spans over some layers, where each layer contains a 2D structure.

Connectivity-sensitive pixel: Consider the topmost layer (let it be *Layer 1*) of Fig. 1(a). As the object is connected, there must exist at least one pixel P_{layer1} which has a 26-neighbor in the layer just below it. We denote such pixels as *connectivity-sensitive* pixels. For the preservation of connectivity, one of these connectivity-sensitive pixels are not interchanged during the first part of the transformation, as the layers present below are hanging from that particular pixel of the top-layer.

Merge Axis: Merge axis ($\mathcal{M}(P)$) is a coordinate axis passing through pixel P and the pixels lying on it are defined to be *non-interchangeable* throughout the first part of the transformation. A merge-axis contains at least one *connectivity-sensitive pixel*. Figure 1(a) shows $\mathcal{M}(P_{layer1})$, $\mathcal{M}(P_{layer2})$ and $\mathcal{M}(P_{layer3})$ in *Layer 1*, *Layer 2* and *Layer 3* respectively. All the pixels of the given 2D component are finally brought onto or *merged* on this axis using connectivity preserving interchanges. Where P is obvious, we use just $\mathcal{M}$.

Level: In a given layer and in a given connected component in that layer, a *level* is the shortest distance of a pixel of that component, from the *Merge Axis*.

Cut and Non-cut pixels: A pixel whose removal disconnects the originally connected component is a cut-pixel, otherwise it is non-cut.

Coordinate Axes: For any black pixel on a 2D layer, its four coordinate axes determine the direction in which the adjacent black pixels are located. The coordinate axes through pixel P in Fig. 1(b) are the following (i) $A(P)_v$ (vertical axis), (ii) $A(P)_h$ (horizontal axis), (iii) $A(P)_{45}$ (making 45° with $A(P)_h$) and (iv) $A(P)_{-45}$ (making -45° with $A(P)_h$).

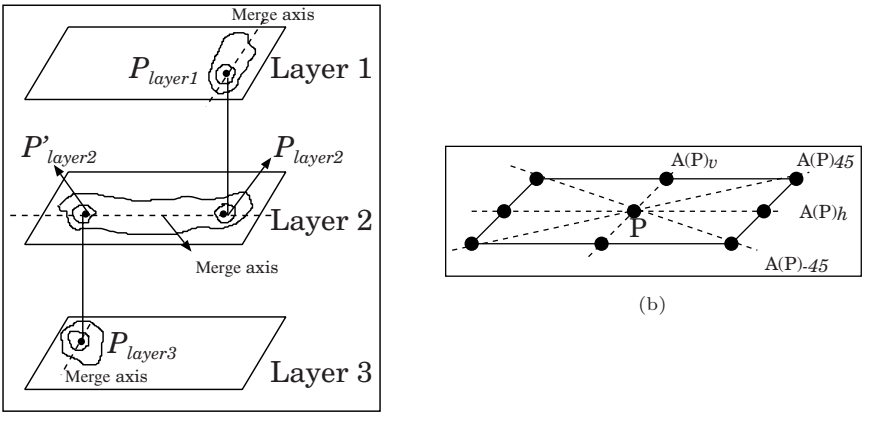


Fig. 1. (a) The 3D object in different layers. The adjacency between P_{layer1} and P_{layer2} (P'_{layer2} and P_{layer3}) maintains the connectivity across *Layer 1* and *Layer 2* (*Layer 2* and *Layer 3*). (b) The coordinate axes through a given P.

Merge Path: Extending the concept of *Merge Axis*, a Merge Path is a path (not necessarily a straight line) on which we finally merge all the pixels.

We use $G = (V, E)$ to denote a graph, where V denotes the set of vertices and E, the set of edges between the vertices. *Complexity Analysis*, wherever used, denotes the number of *interchanges* between black and white voxels required for the particular algorithm.

2.2 Solution Strategy

In this problem, our fundamental strategy is to attack the 2D layers of a $\mathbf{B}_8$-connected finite binary image found in a $\mathbf{B}_{26}$-connected 3D object. In a given 2D layer, a pixel can have at most 8 neighbors. Hence, we borrow from Bose et al. [1] the idea of transforming any 2D binary image into a *vertical* image, ensuring that the object preserves connectivity during the transformation. However, we cannot directly adopt the strategy in [1] since the vertical image produced in the 2D plane is *unique* and in our case might snap the connectivity between two layers. A general case of the problem may have the images I and J such that, each layer has more than one connected component of black pixels.

Define a graph $G = (V, E)$, such that (i) each connected component in any layer corresponds to a node in V, and (ii) for any two connected components, C_1 and C_2, if there is at least one pair of voxels (u, v) which are $\mathbf{B}_{26}$ adjacent, with $u \in C_1$ and $v \in C_2$, then we have an edge. It is easy to see that, as the black pixels in the original image I are connected, G is connected. Let $G' = (V, E')$ be any spanning tree of G. We know from the definition of a spanning tree that, as long as G remains connected G' also remains connected, thus satisfying the principal constraint behind the transformation. So, it is sufficient to consider G', instead of G. Also, we know that every spanning tree has *at least one* node whose degree is equal to one.

3 The Strategy for Voxel Transformation

Before we discuss the actual algorithm we discuss below a construction which is frequently used in the algorithm.

3.1 Construction of 2D Linear Chains

Given a node in G' with degree one, we need to consider only one *connectivity-sensitive pixel* in the component represented by the node to preserve the connectivity. So, a *Merge Axis* can be any *coordinate axis* passing through that pixel. Now, the rest of the black pixels (which do not originally lie on the merge axis) are interchanged preserving the connectivity such that they finally appear as a linearly connected chain along the merge-axis.

The strategy can be outlined as follows. We compress the 2D region, step by step, from the boundary, simultaneously expanding on the merge axis, $\mathcal{M}$.

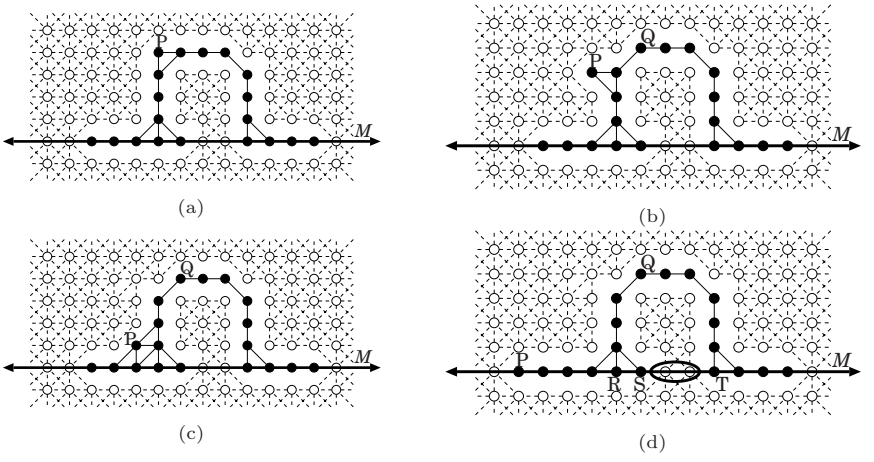


Fig. 2. (a) This shows a connected component in a layer. P is a *non-cut* pixel and M, the *Merge Axis*. (b) This shows P in its new position after the interchange with its adjacent white pixel. (c) P is interchanged with white pixels twice more. (d) P is finally placed on M. This leaves all other pixels on the boundary to be *cut* pixels. Q is one such pixel. The oval region shows the disconnectivity on the *Merge Axis* before collapsing the *cut* pixels.

Ultimately, we have the linear connected chain on $\mathcal{M}$ of all the pixels originally in the 2D plane. Now, we describe our algorithm.

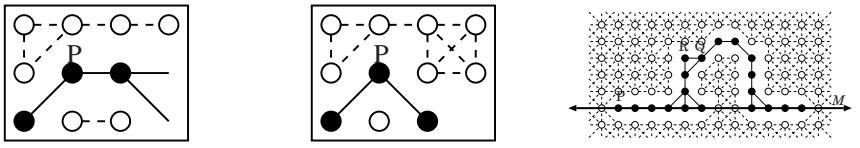
Take a *non-cut* pixel (if any) on the boundary, other than those on $\mathcal{M}$. Clearly, its removal doesn't disconnect the rest of the black region. Hence, we move it along the boundary until we first reach $\mathcal{M}$, interchanging with the white pixels that come in the way. This clearly maintains connectivity of the black pixels. Place it on $\mathcal{M}$, by interchanging with the white pixel already present.

We repeat the above process till all the *non-cut* pixels are exhausted. Now, we are left with only *cut* pixels on the boundary (if any).

Figure 2(a) shows the part of the original image, which is of concern (one layer). The movement of the *non-cut* pixel P along the boundary to the merge axis $\mathcal{M}$ is shown in Fig. 2(b) to Fig. 2(d).

Lemma 1. *Consider the situation when all the black pixels on the boundary, not on $\mathcal{M}$, are* cut *pixels. Also consider a part of the boundary which starts and ends on $\mathcal{M}$ and let m_a and m_b be a pair of black pixels on $\mathcal{M}$ through which the* cut *pixels on this part of the boundary are connected to $\mathcal{M}$. Now, m_a and m_b are connected only through this part of the boundary. Moreover, this is true for every such part of the boundary.*

Proof. Let us suppose that we have another path connecting m_a and m_b. This clearly implies that there is a *non-cut* pixel on the part of the boundary contradicting the hypothesis. The same argument follows for all such parts of the boundary. □



(a) (b) (c)

Fig. 3. (a) One possible location of P, the leftmost pixel on the topmost level. (b) The other possible location of P. (c) The *cut* pixel Q is collapsed onto the previous *level*, exposing a *non-cut* pixel R.

For an illustration, Fig. 2(d) shows a discontinuity on $\mathcal{M}$ with all the black pixels on the boundary and not on $\mathcal{M}$ being *cut* pixels. The pixels S and T are connected only through this boundary.

So, our goal is to fill the gaps between the two ends on $\mathcal{M}$. Consider the leftmost pixel in the topmost level, say P. As this is the topmost level, this pixel has no $\mathbf{B}_8$ neighbors in the level above or to the left of it. Now, considering the remaining possibilities the only two situations where there are no *non-cut* pixels on the boundary are illustrated in Fig. 3. As it is clear from the figure, filling up of the gaps on $\mathcal{M}$ can be clearly done by collapsing P to the *level* below it.

Figure 3(c) shows the collapsing of Q for example. Q is interchanged with the pixel right below it. This is formed from Fig. 2(d). It is easy to see that collapsing preserves connectivity.

We continue collapsing. If this results in a new *non-cut* pixel, we go for the next iteration.

Complexity Analysis: Assume that the total number of pixels in the 2D region is n. Any pixel can be a *cut* or a *non-cut* pixel at any point of time during the transformation. If it is a *non-cut* pixel, and if it is chosen to be moved along the boundary to $\mathcal{M}$, it takes $O(n)$ interchanges to reach $\mathcal{M}$, as the boundary contains at most n pixels. If it is a *cut* pixel, all pixels other than those on $\mathcal{M}$ are *cut* pixels and this particular pixel has been chosen to be collapsed to a *level* below it, then it takes one interchange to do so. There can be at most n such interchanges for any particular pixel. Hence, for any pixel, it takes at most $O(n)$ interchanges and therefore, the complexity is $O(n^2)$.

3.2 Algorithm-Part I

Let u be a node with degree one in G' and also let (u, v) be the edge emerging from u. In other words, the components represented by u and v, say U and V respectively, have at least one $\mathbf{B}_{26}$ adjacent voxel pair (v_u, v_v), with $v_u \in U$ and $v_v \in V$. As the degree of u in G' is one, we develop a strategy to *merge* U with V.

To make the merging easier, we first form a single straight chain of all the pixels on U. The merge axis $\mathcal{M}$ for U can be in any direction. So, let us fix it to be horizontal. We merge all the black pixels in U on $\mathcal{M}$.

Let us suppose that U and V are in a layers i and $i+1$ $(i-1)$, respectively. Again, to make merging easier we take $\mathcal{M}$ to such a location on layer i that the

top view of these two components U and V looks like, $\mathcal{M}$ protruding out from the boundary of V. So, we translate $\mathcal{M}$ horizontally, in the layer in which U is present, say i, till any further move removes the connectivity between U and V, using the procedure described below.

Translation: The idea behind translation is simple. We move pixel by pixel. Figure 4 shows an example of how we do it. The extreme pixel is moved first followed by the next farthest pixel. A given pixel gets displaced $O(n)$ times. There are $O(n)$ pixels to be moved. Hence, the complexity for translation is $O(n^2)$.

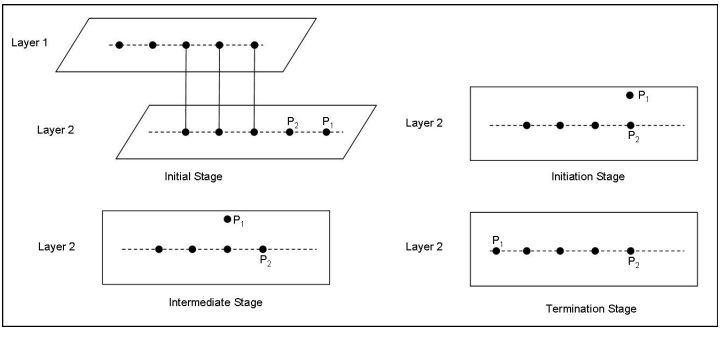


Fig. 4. Here P_1 and P_2 needed to be displaced. In this illustration, the transformation of P_1 is shown. P_1 is moved along the chain (for preserving the connectivity) and brought back to the chain whenever the first white pixel is found. The displacement of P_2 can be similarly done.

If U intersects with any other connected component in the *layer* i either during the process of merging on $\mathcal{M}$ or during the process of translation, we do the following.

1. We stop the process.
2. We consider the compound component formed by U and the component with which it intersects instead of the original components.
3. We build a new G and form the new spanning tree, G'.
4. We go for the next iteration.

Note that, U might have established new links with other components in layers $i-1$ and $i+1$. Figure 5 shows an illustration of this part.

Complexity (Part I): Let n_u and n_v be the number of black pixels in U and V respectively. From our earlier discussions, merging all n_u pixels on $\mathcal{M}$ takes $O(n_u^2)$ interchanges. As translating $\mathcal{M}$ horizontally by one pixel takes $O(n_u)$ interchanges, the entire translation phase takes $O(n_u n_v)$ interchanges. If any process has to be stopped in the middle, then a new iteration has to be started after making some changes, mentioned above.

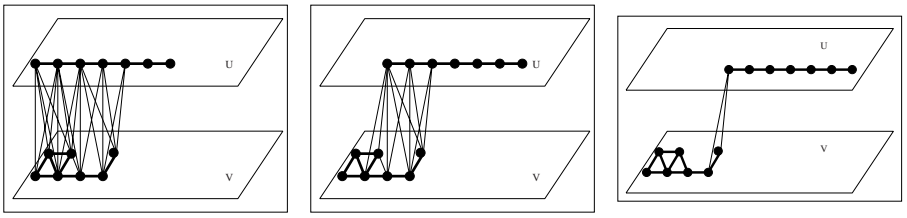


Fig. 5. (a) The pixels in U have been merged on to its *Merge Axis*. This shows all the adjacencies. (b) Two of the pixels have been translated by the procedure described above. (c) The situation after the entire translation.

3.3 Algorithm-Part II

Now, we merge the chain in U with the layer containing V. The pixels can be merged in any order but we restrict to one particular order, namely, from the end of the *Merge Axis* on U which is $\mathbf{B}_{26}$ adjacent to a pixel on V to the other end. There are two possibilities.

Case I: U has developed new $\mathbf{B}_{26}$ adjacencies with some voxels of other components in the layer containing V.
Case II: No such adjacency has been developed.

It is very well possible that some other edge between U and any other component W in G got snapped. Case II is the easiest of the two. All we need to do is, keep interchanging the voxels on $\mathcal{M}$ in U, with the white voxels in the layer containing V starting from either end of $\mathcal{M}$. Figure 6 shows an example. It is easy to see that the complexity in this case is $O(n_u)$. Now, let us consider Case I. Then, there is a possibility that, if we follow the same steps as suggested above for Case II, after certain number of steps, we encounter another connected component. Figure 7 shows an example. If we encounter such a component, we adopt a sequence of steps, similar to those considered in *Part I*.

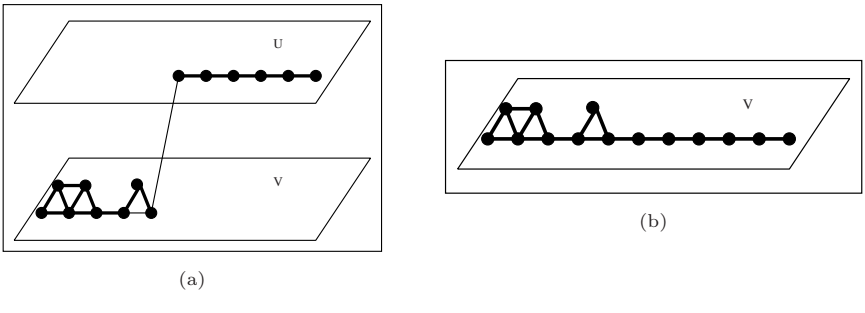


Fig. 6. (a) Starting from the situation in Fig. 5(c) a pixel has been merged with V. (b) All pixels on U have been merged onto V.

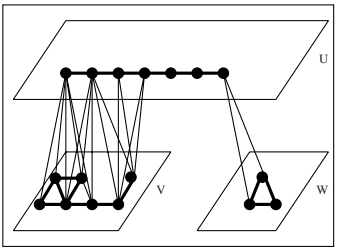


Fig. 7. U develops new adjacencies with W, during translation

1. We stop the process. This may be the end of the process.
2. We consider the compound component formed by V and the other component in the same layer which we encountered, along with the pixels interchanged between U and this layer by now, instead of V and that other component.
3. We update U. U now contains fewer pixels on the chain $\mathcal{M}$.
4. We rebuild G and form the new spanning tree, G'.
5. We start a new iteration.

3.4 Proof of Correctness and Overall Complexity

Lemma 2. *The algorithm suggested above, eventually leads us to the intermediate structure, a single chain containing all the black voxels in the original image I.*

Proof. In one pass through *Part I* of the algorithm, we either merge U with another connected component in the same layer, i, or move the chain, $\mathcal{M}$ to a new location, again in the same layer, i. So, decrease in $|V|$ in *Part I* is less than or equal to one. In one pass through *Part II* of the algorithm, we merge either U with V or V with some other connected component in its layer or both. So, decrease in $|V|$ is either one or two.

Now, if any pass through *Part I* merges two connected components, we don't touch *Part II* until again we pass through *Part I*, as a new iteration is started. If the pass through *Part I* doesn't merge but, simply translates U to a new location, we definitely pass through *Part II* and this guarantees that at least two components will be merged. Hence, each iteration through the algorithm reduces $|V|$ by at least one and therefore, after at most $|V|-1$ iterations, we are left with a single component. Now, we can form $\mathcal{M}$ for this single component in any direction starting from anywhere and form the single chain. □

Let us find the complexity of an iteration. Assume that *component* i has n_i number of black pixels and that there are c components in total. Note that, components in a particular layer may be disconnected but they can be connected using voxels of layers above and below. Let $\sum_{i=1}^{c} n_i = n$. We divide the complexity calculation into two parts as follows.

1. Merging of all the pixels in a single component.
2. Merging of different components.

Merging a component of n_i black pixels onto its *Merge Axis* takes $O(n_i^2)$ interchanges. Now, during the process, if this intersects with another component with n_j number of black pixels, we simply start a new iteration. Let n_k be the total number of pixels of the component on which this *Merge Axis* has to be merged. Translation of the *Merge Axis* takes $O(n_i n_k)$ interchanges if it doesn't intersect with any other component. Else, we simply start a new iteration. Once translation is done, merging takes $O(n_i)$ interchanges if it's Case II. In Case I, we have to start a new iteration somewhere in the middle.

So, the worst case complexity of an iteration is

$$O(n_i^2) + O(n_i n_k) + O(n_i) = O(n_i^2 + n_i n_k)$$

And the overall worst case complexity is simply a summation of the above complexity over all the iterations. From Lemma 2 it is clear that the number of iterations is at most $c-1$. Note that n_i, n_k may change after every iteration due to merging of components. In any case, n_i and n_k are $O(n)$. So, an upper bound of the complexity is

$$\Sigma_{i,k=1}^{c-1} O(n_i^2 + n_i n_k) = \Sigma_1^{c-1} O(n^2) = O(cn^2)$$

Theorem 1. *Given any two binary images I and J with c_1 and c_2 number of connected components respectively and n voxels each, both can be transformed into one another maintaining the original connectivity of the black voxels by $O((c_1 + c_2)n^2)$ interchanges.*

Proof. The theorem follows from Lemma 2 and the above discussion. □

4 Voxel Transformation under a Different Connectivity Model

In the model presented till now, a *valid interchange* is taken as such an interchange between any two $\mathbf{B}_{26}$ adjacent black and white voxels, which preserves the connectivity of the image *before* and *after* the interchange. A slightly different model can be obtained if we impose a *single backbone* condition [2] along with our original connectivity model. Dumitrescu and Pach [3] also consider this as an alternative model. In our case, a *backbone* is defined as the set of all black voxels except the one which we currently interchange. The condition is that the *backbone* must be $\mathbf{B}_{26}$ connected at any given point of time. In order to adopt this model, we only need to make small changes in our algorithm.

First, note that, in the algorithm we described, there are only two situations where the *single backbone* condition fails.

1. While collapsing the pixels on the boundary when all the *non-cut* pixels not on *Merge Axis* are exhausted to form the 2D linearly connected chains.
2. While merging the translated *Merge Axis* with a component in an adjacent layer.

First Situation: While forming the 2D linearly connected chains, instead of collapsing the pixels to the *level* below when all the pixels on the boundary are *cut*, we can move the black pixels between m_a and m_b on the *Merge Axis* (refer Lemma 1) to one of the extreme ends of the axis (through interchanges). This is similar to the *Translation*, mentioned in *Part I* in Section 3.2. So, ultimately what we have is a *Merge Path* which is the union of two parts of the original *Merge Axis* and the *cut* black pixels on the boundary. For example, consider Fig. 2(d). The pixels R, S and T have to be translated to the ends of M.

This can be easily adopted to the algorithm discussed. We need to consider only one *connectivity-sensitive pixel* for each 2D connected component. So, we can easily decide which part of the *Merge Axis* is to be extended and which part should be left untouched (depending on which part the *connectivity-sensitive pixel* lies on). For example, suppose that in Fig. 2(d), the pixel R is the *connectivity-sensitive pixel*. We should not move R during the translation mentioned above. So, a possible and easy solution is to translate all the pixels starting from the rightmost end of M till T to the left end of M. Then, translate S. And we end up with the *Merge Path*.

Now, we are left with *bending* the *Merge Path* to a straight line. The only curvy portion is that of the chain of black pixels. Again, in a similar manner, considering the above example, translate each pixel on the chain, starting from the right end to the left end of M.

Second Situation: We only need to change the order in which the pixels on the *Merge Axis* are merged with the component in the other layer. Note that the end of the *Merge Axis* other than the one whose removal disconnects the components, is a *non-cut* pixel. So, we can merge starting from that end, just the opposite way we mentioned in Section 3.3. Now, clearly, interchanging with a *non-cut* pixel maintains the backbone's connectivity. The only problem is with Case I of the Section 3.3. The new adjacencies are at the very end where we have *non-cut* pixels. One possible solution is starting from this end, find the first pixel which is not $\mathbf{B}_{26}$ adjacent with any of the pixels in the new component which the Case I refers to. So, starting from this pixel (which is clearly *non-cut*) keep merging till the other end. The rest of the algorithm follows.

The changes mentioned in both the above mentioned situations do not change the complexity.

5 Conclusion

To conclude, we have shown that two 3D binary images I and J of c_1 and c_2 components differ by a sequence of $O((c_1 + c_2)n^2)$ 26-local interchanges preserving the original black connectivity. We also discussed an alternative approach to fit into a slightly different model in Section 4. One possible extension to the algorithm would be to consider a more general model where we try to preserve the connectivity of the white pixels (background) along with that of the black voxels (foreground).

References

1. Bose, P., Dujmovic, V., Hurtado, F., Morin, P.: Connectivity-Preserving Transformations of Binary Images. In: Computer Vision and Image Understanding, Elsevier, Amsterdam (accepted, 2007)
2. Dumitrescu, A., Suzuki, I., Yamashita, M.: Motion planning for metamorphic systems: feasibility, decidability and distributed reconfiguration. IEEE Transactions on Robotics and Automation 20(3), 409–418 (2004)
3. Dumitrescu, A., Pach, J.: Pushing squares around. Graphs and Combinatorics 22(1), 37–50 (2006)
4. Klette, R. and Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufman, Elsevier, New Delhi, India, (2005)
5. Rosenfeld, A., Saha, P.K., Nakamura, A.: Interchangeable pairs of pixels in digital images. Pattern Recognition 35(9), 1853–1865 (2001)
6. Rosenfeld, A., Nakamura, A.: Two simply connected sets that have the same area are IP-equivalent. Pattern Recognition 34(2), 537–541 (2002)