

Solutions to Homework 1

1. Morphological operators on a hexagonal lattice:

(1) $S=\{1\}$, $L(a,b)=b$, $n=1$, $f=1$ goes with (M) Reproduces the input image without change (identity operator)

Since $L(a,b)=b$, it doesn't matter whether surround 1 is found or not. The output is just equal to the input b .

(2) $S=\{\}$, $L(a,b)=ab$, $n=1$, $f=1$ goes with (I) Resets all cells to zero

Since S is the empty set, $a=0$ always.

(3) $S=\{1\}$, $L(a,b)=ab$, $n=1$, $f=1$ goes with (C) Keeps only isolated cells that are one.

The output is equal to 1 only if the input pixel b is equal to one AND the surround is all zeros.

(4) $S=\{2\}$, $L(a,b)=b\bar{a}$, $n=1$, $f=1$ goes with (L) Removes ends of lines

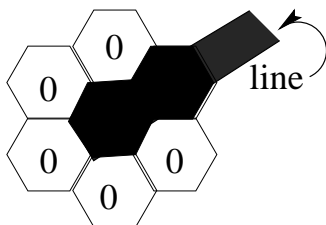
Here the output pixel is equal to zero if the input pixel is a zero. In addition, the output pixel is equal to zero if the input pixel was a 1, and if also exactly one of its neighbors is one. In this case, it can be considered the end of a line, and it is getting removed. See the figure below left.

(5) $S=\{7\}$, $L(a,b)=ab$, $n=1$, $f=1$ goes with (D): Removes edges of blobs, keeping the interior

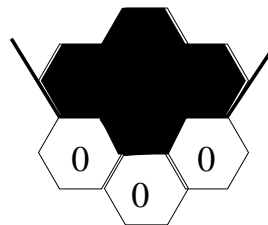
If the input pixel is a zero, then it stays a zero. But if it's a one, it gets converted to a zero UNLESS all the neighbors are one. That means a pixel can stay equal to one only if it is in the interior, surrounded on all sides by ones.

(6) $S=\{4\}$, $L(a,b) = ab$, $n=1$, $f=1$ goes with (P): Marks all corners, flushes the rest

Here the output is equal to one only if the input pixel is one, AND also the surround consists of 3 adjacent ones on one side, and 3 adjacent zeros on the other side. This particular pattern can be considered a corner (see the figure below right), in which case the "corner" pixel gets marked with a 1, and anything else gets zero.



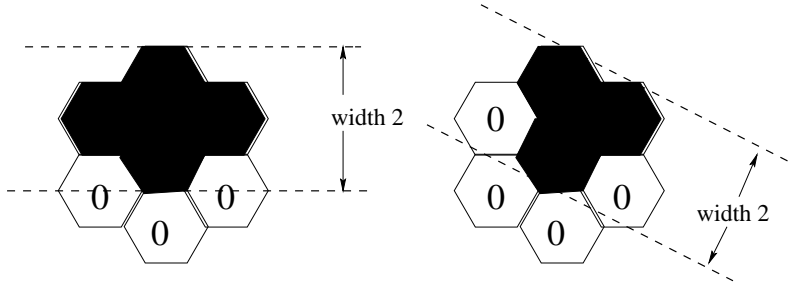
End of a line.



A corner

(7) $S=\{3,4\}$, $L(a,b) = \bar{a}b$, $n=\infty$, $f=3$ goes with (N): Skeletonizes until lines are only one picture cell wide

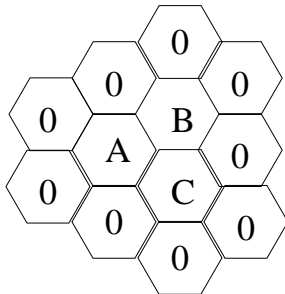
The output is a 1 only if the input pixel is a 1 and also the surround is not 3 or 4. Any time the center pixel is equal to one, and the surround is either 3 or 4, we have a situation where the picture item is greater than one cell wide:



By removing that center cell (changing it from a one to a zero) we are removing the double width. Of course, there are other patterns as well that have double width. For example, pattern 5 with the center cell equal to 1. But here we can expect that there would be other cells removed first. The lower left cell of surround 5, for example, may be the center cell of surround 3, and would therefore get removed that way. Subsequently, the remaining pattern would look like a surround 4.

Since we have $n = \infty$, the process is continued until the picture item is down to a one-cell-wide skeleton.

The reason for the 3-fold tessellation is that we do not want to wipe out any patterns entirely. Suppose we have something that looks like this:



where all the pixels A,B,C are equal to 1. If they are all processed simultaneously, then each of the three will get erased, as each has a surround that is a (rotated) version of surround 3. If one of them goes first however, and gets removed, then the others will not get removed, since they no longer find themselves with surround 3.

(8) $S=\{1,2,3\}$, $L(a,b) = \bar{a}b$, $n=\infty$, $f=1$ goes with (G): Cuts off all appendages (that is, thin lines) and removes tiny blobs

In this case, the output is 1 only if the input is 1 and also the surround is not one of 1,2,3. Surround 1 means we have an isolated 1-cell; this gets removed. Surround 2 means we have the end of a line; this gets removed. Surround 3 means we have a small appendage protruding from a larger item; this also gets removed. The smallest item which does not get eaten away is a compact blob of 7 one's (central cell and its 6 neighbors). Several people thought this

operation was nearly the same thing as (g): removing edges. But in fact, this operator does not remove (at all) the edges of a large blob.

(9) $S=\{5,6,7\}$, $L(a,b) = a + \bar{a}b$, $n=\infty$, $f=3$ goes with (A): Fills in small cavities

Now we have a function L which is going to increase the number of 1-cells. The output will be one any time the input is one OR the surround is one of 5,6,7. But 5,6,7 are all cases where there is an interior hole (zero) or a “bay” of zeros. These holes and cavities get turned into ones, “filled in.”

(10) $S=\{8\}$, $L(a,b) = ab$, $n=1$, $f=1$ goes with (H) Mark all places where 3 thin lines come together

If 3 non-touching single-pixel lines comes together in one pixel, it would look like surround 8 surrounding a 1.

(11) $S=\{1\}$, $L(a,b) = ab$, $n=1$, $f=1$ followed by $S=\{7\}$, $L(a,b) = a\bar{b}$, $n=1$, $f=1$ goes with (I) Resets all cells to zeros.

The first operation is the same as (3) above. Only isolated cells that are one are kept. The second operation would mark with a 1 any place where there is an isolated zero found surrounded by ones. However, after the first operation, there won't be any occurrences of these to find with the 2nd operation. So, in the second operation, $a=0$ always, and the output is zero.

(12) $S=\{7\}$, $L(a,b) = a\bar{b}$, $n=1$, $f=1$, followed by $S=\{1\}$, $L(a,b)=ab$, $n=1$, $f=1$ goes with (Q) Marks any isolated zero cell

The first operation will mark any isolated zero cell. Marking it means putting a 1 there. The second operation, as before, would retain any isolated one cell, but after the first operation is finished, the only isolated ones appearing will be the things that got marked during the first operation, that is, the isolated zero cells.

(13) $S=\{1,7\}$, $L(a,b)=ab$, $n=1$, $f=3$ goes with (K) Removes edges of blobs, keeping only the interior, but making sure not to completely erase any blob

This is similar to (5) above, in that edges are being erased. But since we also put surround 1 in the set, any isolated pixel will be kept. Since we put in 3-fold tessellation, no blob can get wiped out completely.

(14) $S=\{1\}$, $L(a,b)=ab$, $n=5$, $f=1$ goes with (C) Keeps only isolated cells that are one.

With $n=1$, this is the same as (3) above. Isolated ones are kept. After we do it once, there are only isolated ones left in the picture. Doing it 4 more times doesn't change anything.

(15) $S=\{7\}$, $L(a,b)=ab$, $n=5$, $f=1$ goes with (E) Removes 5 layers of edge pixels from blobs (for example, any blob whose maximum dimension is 10 pixels or less will be erased)

This is like the erosion operator in (5) above, except we are iterating on it 5 times.

2. Binarization by thresholding:

If you tweak the threshold one way, then more pixels will be black. This is good as far as filling in the holes inside the worm body, but it is bad as far as getting more stray pixels on the outside of the body. If you tweak the threshold the other way, you get fewer black pixels, so again you improve certain things and make others worse. There is no threshold that is perfect, which is why we are going to have to clean things up with opening and closing operations.

3. Dilating and eroding:

Let's try eroding first.

```
a = bwmorph(b,'erode');
```

Grows the black dots. Inside nicely filled in, but dots too big now.

```
a2 = bwmorph(a,'dilate');
```

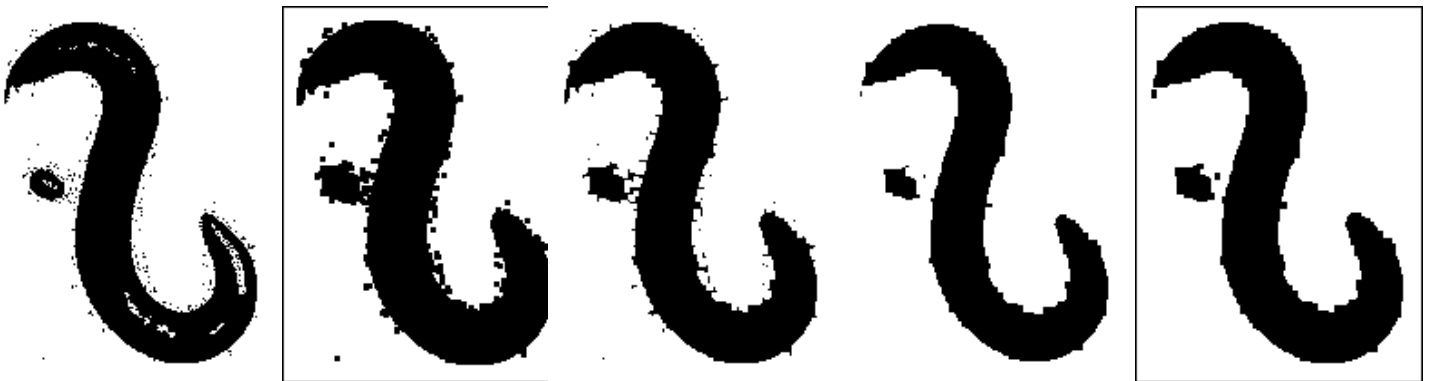
Better, but still a lot of messy stuff

```
a3 = bwmorph(a2,'dilate');
```

Better, but still not very good.

```
a4 = bwmorph(a3,'erode');
```

Should now be back to being about the right size.



Here are the five images, in the order: b, a, a2, a3, a4. That helped somewhat, but the edges are still messy (that is, there are protruding black pixels). What if we dilated first?

```
d = bwmorph(b,'dilate');
```

Nicely cleans up ALL the bad dots outside

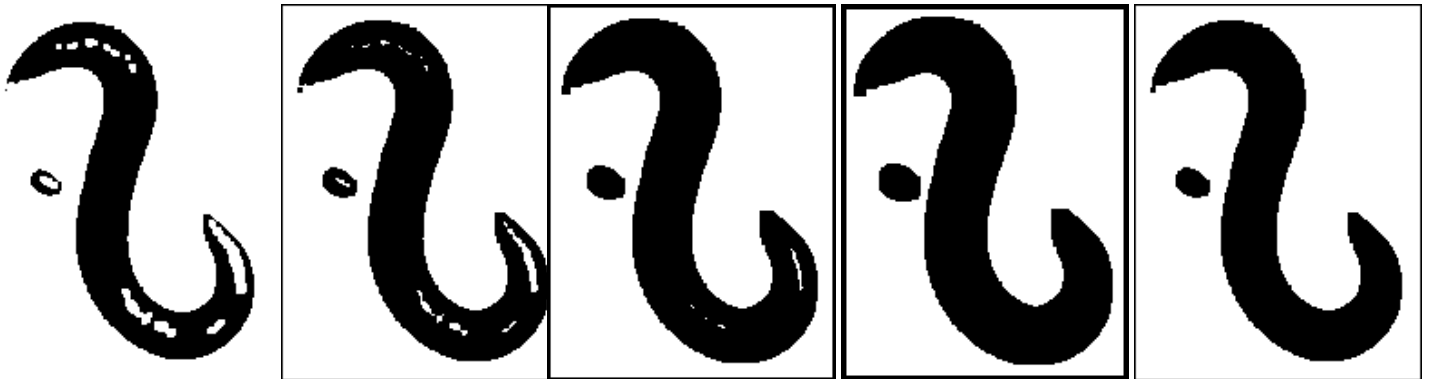
```
d2 = bwmorph(d,'erode');
```

Back to the right size

```
d3 = bwmorph(d2,'erode');  
Holes are not quite filled in yet, need to erode again
```

```
d4 = bwmorph(d3,'erode');  
Perfectly filled in!
```

```
d5 = bwmorph(d4,'dilate');  
d6 = bwmorph(d5,'dilate');  
Gets it back to the right size.
```



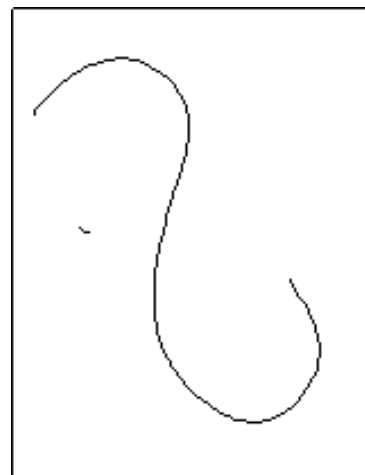
Here are five of the images, in the order: d, d2, d3, d4, d6. So dilating first is better than eroding first. Our sequence of commands is: dilate, erode, erode, erode, dilate, dilate.

4. Skeletons:

If we say: `d6skel = bwmorph(d6,'thin',Inf);`
then we get a strange picture, because we are getting the skeleton of the background.
Remembering that object pixels need to be value 1 for Matlab, we need to do:

```
d6skel = bwmorph(~d6,'thin',Inf);
```

This skeleton is shown at the right.
`imshow(~d6skel)`



Using `bwlabel`, we find that the worm skeleton is 301 pixels long. The egg skeleton is only 5 pixels long.

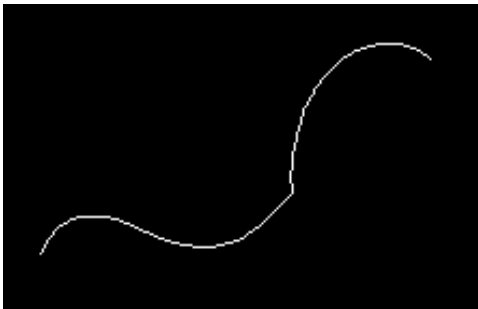
5. Spur removal, the easy way:

Now we consider the image `worm2.tif`. Using the same sequence of commands as before for binarization, dilation, erosion, and skeletonization, we get the following clean binary version and skeleton:



First we extract the inner rectangle that contains the skeleton of interest. `small = worm2d6skel(2:144,2:144)`. By trial and error, we find that the smallest number of iterations that removes the spur entirely is 13:

```
s1 = bwmorph(small, 'spur', 13);  
imshow(s1)
```



6. Effect of the spur:

Does this skeleton look the same as what we would have gotten if the egg had been perfectly erased from the original grayscale image? No. There is clearly a little “jog” near the middle of the skeleton, where the skeleton got pulled over because of the presence of the egg. Even after the spur is removed, the jog is still there. If the egg had been perfectly erased to start with, the skeleton would have been a smoother curve.

7. Sensitivity to the number of iterations:

If we had chosen a much larger number of iterations, such as 40, for the spur removals, the end result would have been exactly the same. Once the algorithm has enough iterations to eat away the spur, additional iterations would not affect the region where the spur used to be. Additional iterations would just continue to eat away at the legitimate end points. But they can be perfectly re-grown. The only problem is when the number of iterations gets so large that the legitimate end points get eaten away back to the base of the spur, in which case the re-growing operation will re-grow the spur as well. So the algorithm, in this particular case, is not sensitive to the exact value of this parameter.