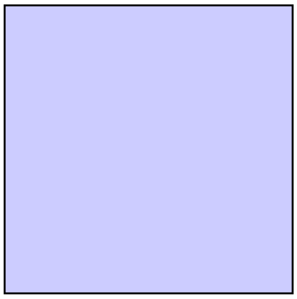
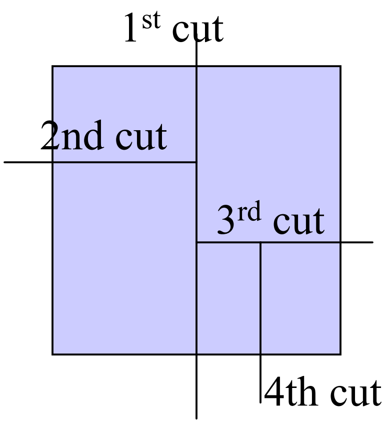
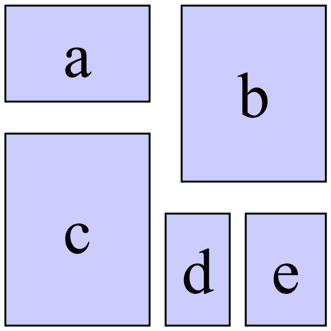
Homework 4: Review of Floorplanning Problem 18-760 Fall 2001



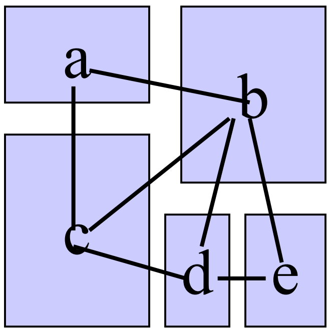
1. Get some graph paper, draw a big square.



2. Draw some "cut" lines thru the square, recursively like this.



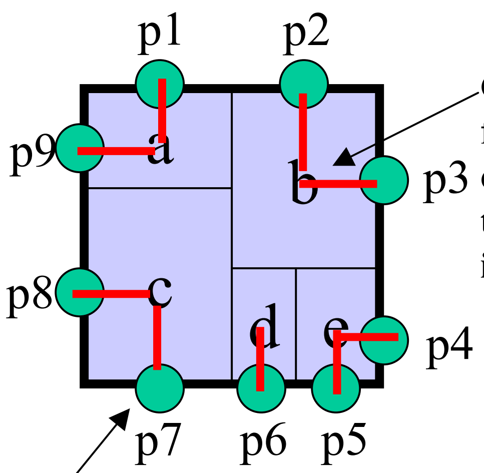
3. This is your trivial "jigsaw" puzzle, and you know it can be reassembled into a perfect square. Label the pieces.



4. Draw "wires" from center of each piece to other pieces it touches. Note – not all pieces will end up with these wire, since not all pieces touch each other

	a	b	c	d	e
a		1	1		
b	1		1	1	1
c	1	1		1	
d		1	1		1
e		1		1	

Make a matrix C(i,j) that records which blocks have wires to which other blocks

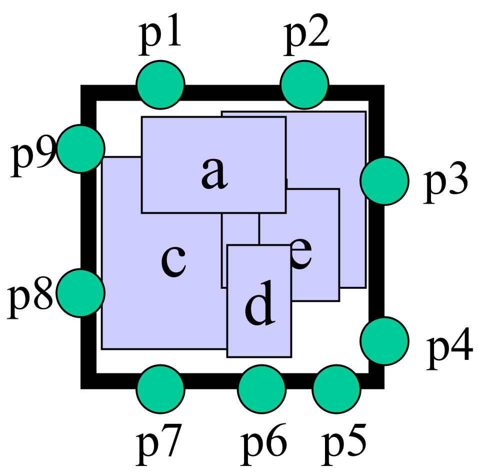


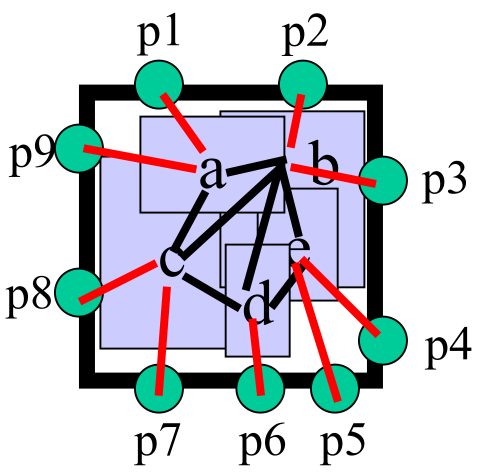
Create another wire from each pin to the center of the block it touches. Note, each pin is on just one wire.

p1 (x1,y1,a)
p2 (x2,y2,b)
p3 (x3,y3,b)
p4 (x4,y4,e)
p5 (x5,y5,e)
p6 (x6,y6,d)
p7 (x7,y7,c)
p8 (x8,y8,c)
p9 (x9,y9,a)

5. With the puzzle "assembled", draw a "pin" on the perimeter with one same coord as the center of each block that touches the perimeter. In this small example, its all the blocks. It general, it won't be all the blocks.

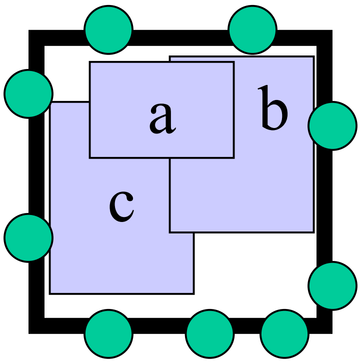
6. Create an array that lists each pin, its FIXED location on the perimeter of the square, and what block it connects to.

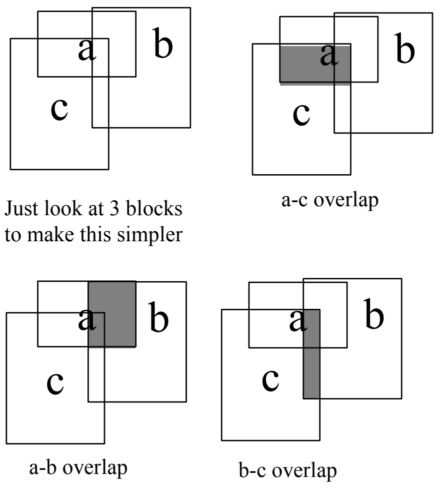




7. To anneal: just randomly relocate a block inside the overall square; these are your moves.
Yes—just let them overlap!
Your problem state is just the (X,Y) center location of each block.

8. The first part of your cost function is just the lengths of all the wires: The ones block-to-block, and the ones pin-to-block. Remember: blocks move, but pins don't.





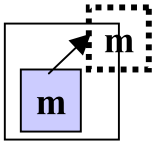
9. The second part of your cost function encourages the blocks NOT to overlap.
This is called a "penalty function".
For each pair of blocks that overlap, you add to the cost function the amount
***(area of overlap)*^{2}**

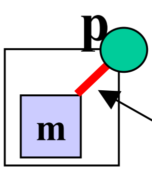
So, overall cost function is this:

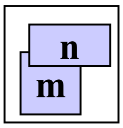
$$\text{Cost} = \sum_{\text{pins } P} \text{distancePintoBlockCenter(pin P , block m)}$$
$$+ \sum_{n} \sum_{m>n} C(m,n)\ \text{distanceBetweenCenters(block n, block m)}$$
$$+ W \sum_{n} \sum_{m>n} \text{overlapArea(block n, block m)}^2$$

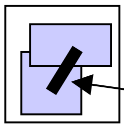
You need this since the units of the first 2 terms are "wirelength", but units of this term are "area". You need to empirically balance these to get a good final solution.

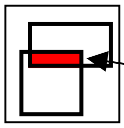
Here is the overall pseudocode for how to do this move-one-block-and-eval to anneal this problem:

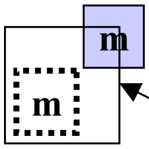


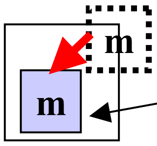








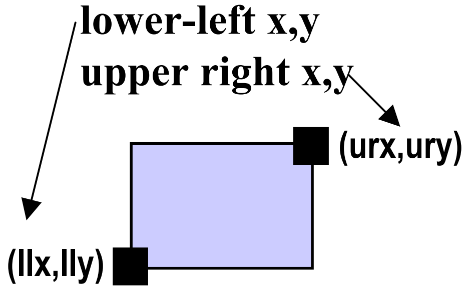




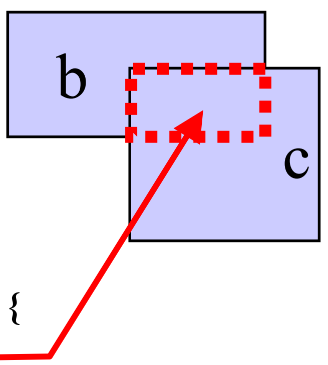
```
// to do a move
let m = random block in {a,b,c,d,e}
let x,y be a random new center location inside the overall square
move block m to location (x,y)
// you can either let the block move partially “outside” the square
// or check when this happens, and slide it back inside.  Your call.

// to eval delta-cost for a move
newCost = 0;

// cost for wires from pins to blocks
for (pin p = 1; p<=9; p++)
   newCost += distanceFromPinToItsBlockCenter(p)

// cost of wires and overlaps block to block
for( block m = a;  m <=e;  m++) {
   for(block n = m+1; n<e;  n++) {

      // cost for block to block wires
      if (C[m][n] == 1)  //blocks m,n have a wire we must count
         newCost += distanceBetweenBlockCenters(m,n)

      // penalty for block to block overlap
      newCost +=  W *overlapAreaBlocktoBlock(m,n)**2
   }
}

// compute final change in the cost value due to this move
deltaCost = oldCost – newCost;

// usual Metropolis accept/ reject
if ( Metropolis criterion(deltaCost, temperature) == “accept” )
      // this is your new state of the floorplan
else
      // nope, sorry. Remember to UNDO this move
      move block m back to its original location
```

Also, its not too hard to calculate the overlaps between the rectangles. You can use this pseudo code.

A rectangle is 4 coords:
lower-left x,y
upper right x,y



Want to compute how much rectangles b,c overlap, area of this potential overlap, (=0 of no overlap)



```
Overlap(rectangle b, rectangle c) {
  rectangle a;
  // try to build the overlap rectangle itself – call it "a"
  a.llx = Max( b.llx, c.llx );
  a.urx = Min( b.urx, c.urx );
  a.lly = Max( b.lly, c.lly );
  a.ury = Min( b.ury, c.ury );

  if(  (a.llx > a.urx) || ( a.lly > a.ury) ) {
    // they don't really overlap
    return (0);
  }
  else {
    // they really do overlap, compute area of rect a, return it
    return (  (a.urx - a.llx) * (a.ury - a.lly) )
  }
}
```