FIXED-OUTLINE FLOORPLANNING WITH RECTILINEAR SOFT BLOCK

109511135張祐宸、109511159李國維

PROBLEM DESCRIPTION

Given

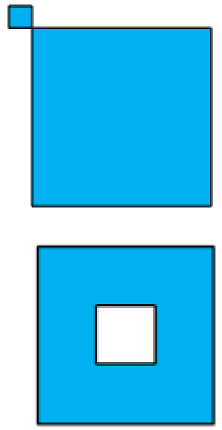
- Fixed outline
- Preplaced fixed module
- Soft module and its minimum legal area
- Soft module shape constraint

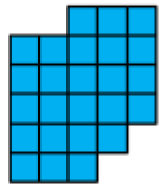
Goal

- Legal placement and minimize wirelength

Soft Module shape constraint

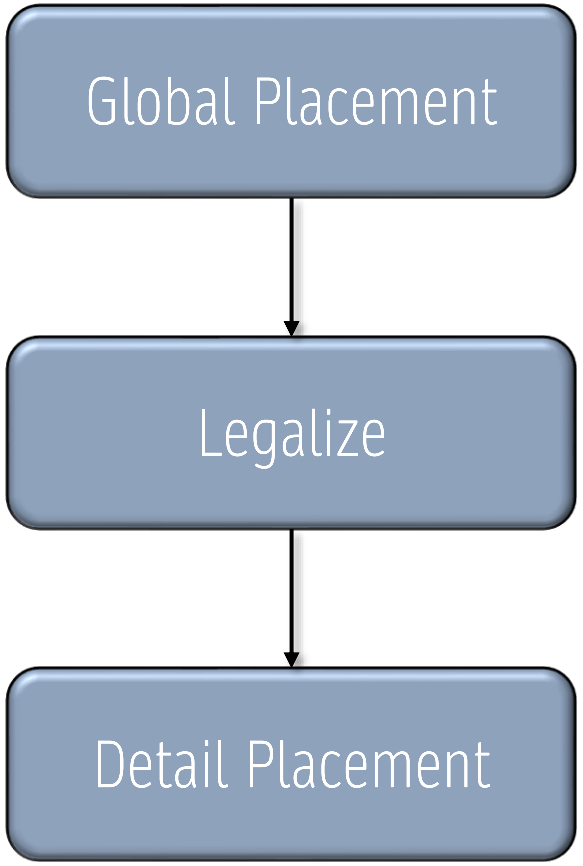
- Simple Rectilinear Polygon
  - All sides are parallel to the axes
  - All sides should be intersect-free
  - Only one area is surrounded by the polygon and no hole inside the polygon
- Minimum area
- Aspect ratio: 0.5~2
- Rectangle Ratio



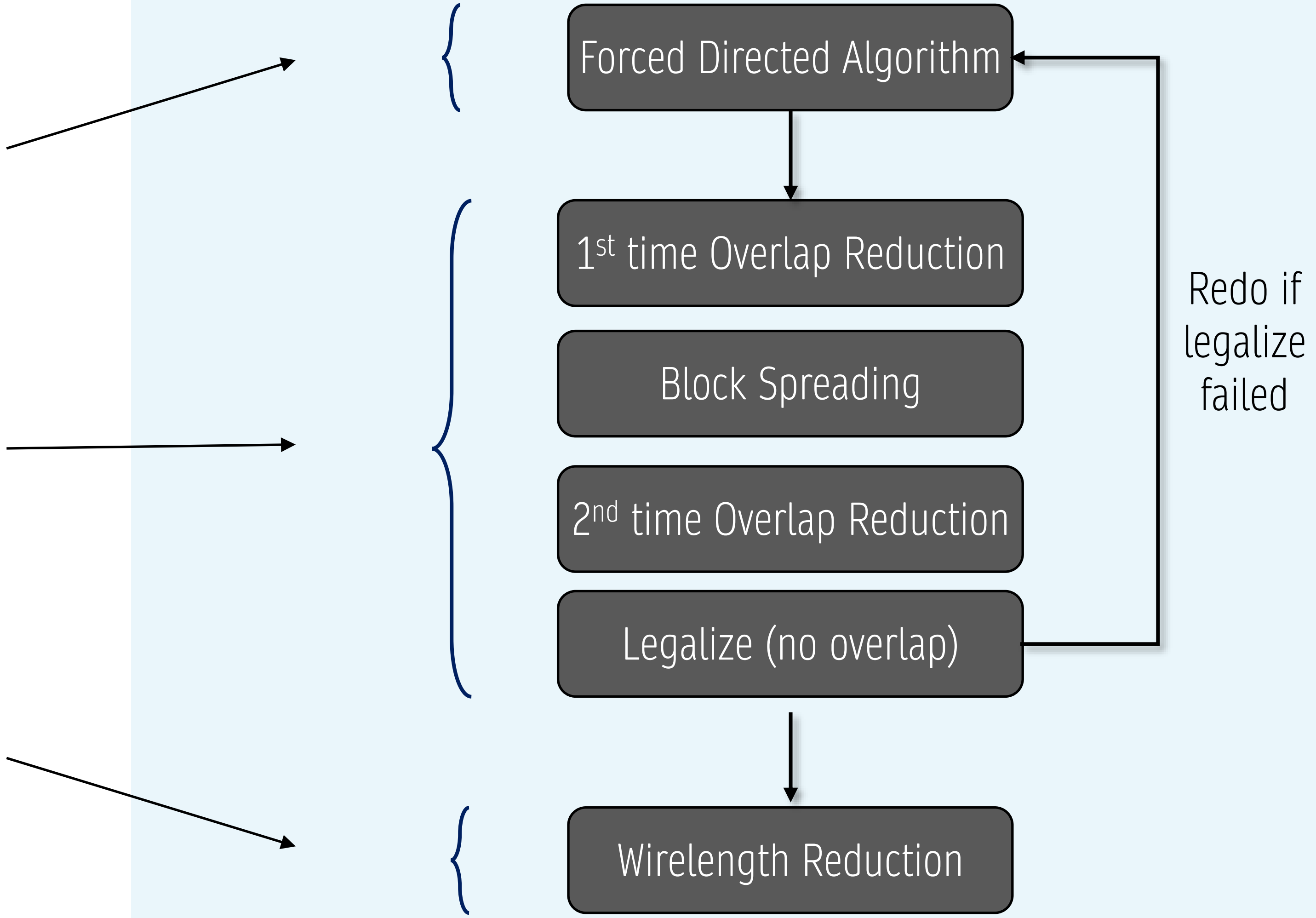


Minimum area required: 20
Actual Area: 25
Aspect Ratio: 1.2 (6/5)
Rectangle Ratio: 83.3% (25/30)

Program Flow



DETAIL PROGRAM FLOW



FORCED DIRECTED

- Reduce the placement problem to solving a set of simultaneous linear equations to determine equilibrium locations for cells.

Flow:

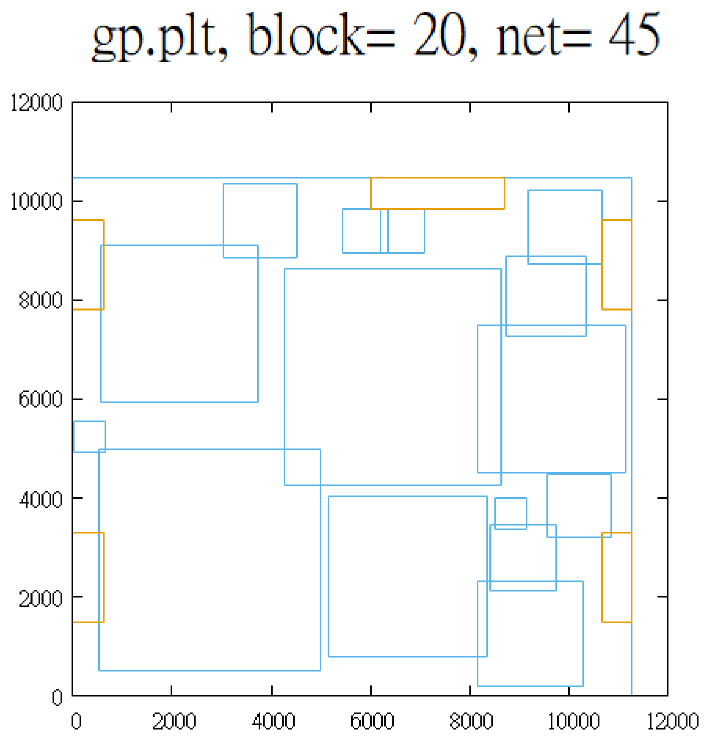
- Start with an initial placement (All in the middle in our case), fixed blocks are locked.
- Sort the blocks in decreasing order for their connectivity in a list (Multimap is used).
- While the Iter_count < Iter_limit (3 in our case)
    - Pull one soft block out of the list.
    - While End_Ripple is false
        - Compute the zero-force location for the cell.
        1. If the location is vacant, lock the block, End_Ripple = true, Abort_Count = 0.
        2. If the location is the same as previous location, End_Ripple = true, Abort_Count = 0.
        3. If the location is occupied and locked, move the cell to the nearest vacant location, End_Ripple = true, Abort_Count += 1.
            - If Abort_Count > Abort_Limit (10 in our case), unlock all cell, Iter_count += 1.
        4. If the location is occupied but not locked, select the occupied block as the block for the next move, exchange the block at that location and lock it. End_Ripple = false, Abort_Count = 0
- End

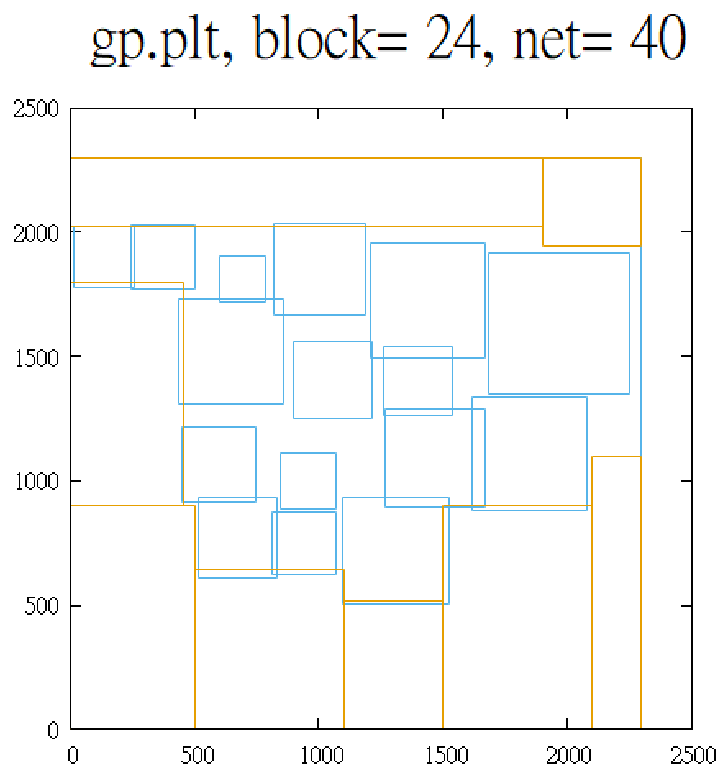
CONT'D

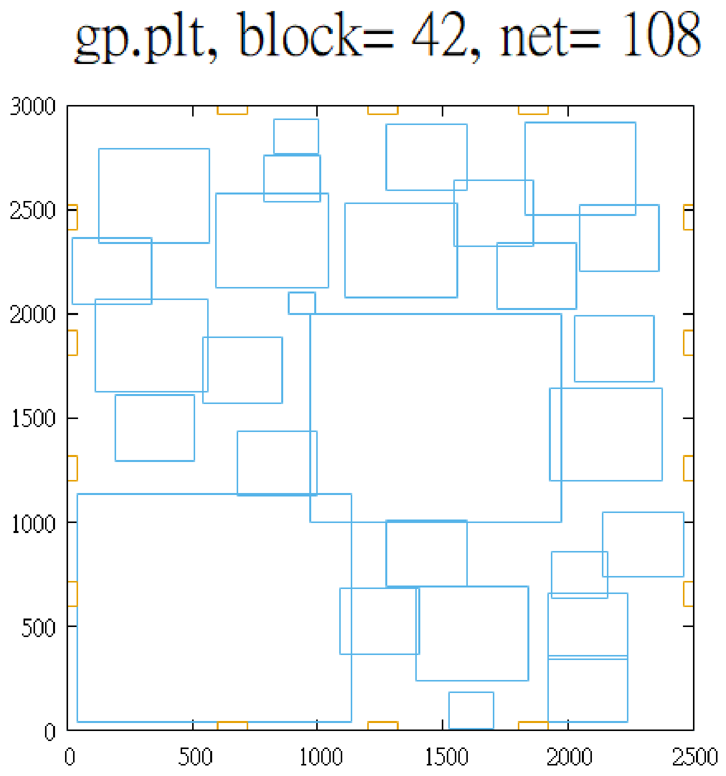
Problems:

1. **How is the shape of the soft blocks are determined?**
   - Sqrt of n must >= minimum area, then start decreasing one side until there's a minimum excess of area
2. **What if the zero-force location is vacant or occupied but there exist blocks that are near that location that cause excessive overlap area?**
   - We set a constraint that overlap length cannot be greater than 1/6 of both the width or height for both the block ("find_if_occupied" function).
   - If the constraint cannot be met, we treat it like there's an occupied block that are locked, a new location need to be found ("Update_location" function)
3. **How does "Update_location" function work?**
   - We first do translation on both axes and try to find a legal location (vacant and met overlap constraint)
   - If translation failed to find a legal location, we do 1000 times random generated coordinates to try to find a legal location. If we failed over 1000 times, we neglect the all constraints as long as the whole block remains inside the chip boundary.
4. **What if the location has an occupied block that's not locked, we found out that the block exceed the chip boundary after exchanging because of the different sizes of block?**
   - We translate until the whole block is in the boundary, neglect all other constraint for the moment.
5. **Locations that met the constraint can be hard to find with large blocks**, so we first find all the blocks that have area >= 1/5 of the largest block's area and do the algorithm first for them, locked them after the algorithm finish. Then run the algorithm for the rest while treating those large blocks as fixed blocks.
6. **What if there are no connections between fixed blocks and soft blocks?**
   - If there are no connections between fixed blocks and soft blocks, the algorithm won't work since every soft blocks starts in the middle. We add one connection between fixed and soft blocks to let the algorithm run normally, we'll delete it when algorithm ends.

EXAMPLE







Note:

- Forced Directed algorithm is done 10 times, the one with the least amount of overlapping area is chosen to continue the rest of the program.

OVERLAP REDUCTION

- Forced Directed Algorithm allowed overlap, we first do one run of overlap reduction algorithm to reduce overlap area, another run is done after block spreading.

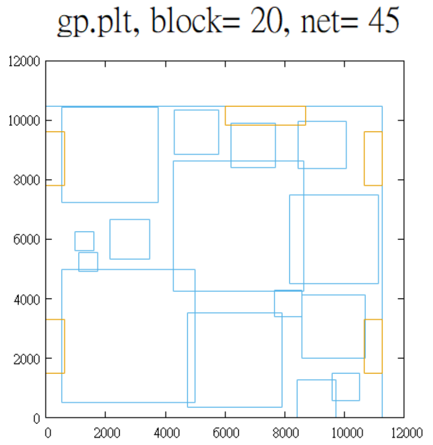
Flow:

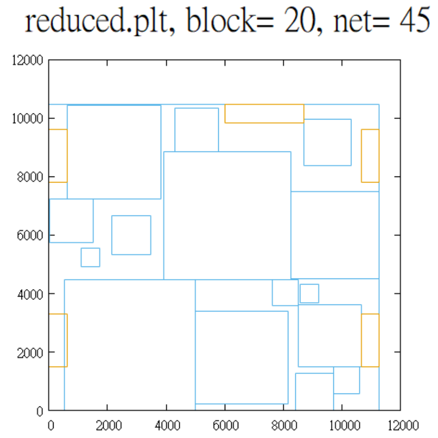
- Check if the overlapping area is reduced if we move the block to right or left.
    - If any of the direction reduces the overlapping area, we move toward that direction until overlapping area is greater than the previous move.
- Do the same thing with Up and Down direction.
- Loop the previous two step until there is no more improvements.
- We found those blocks that have over 20% of overlapping area then random generate coordinate for them, move them to the new location if the overlapping area reduces. This is repeated 100k times.

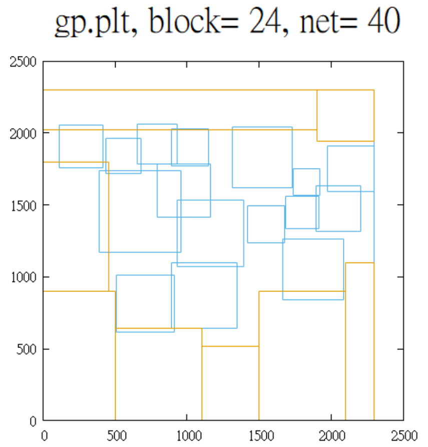
Note:

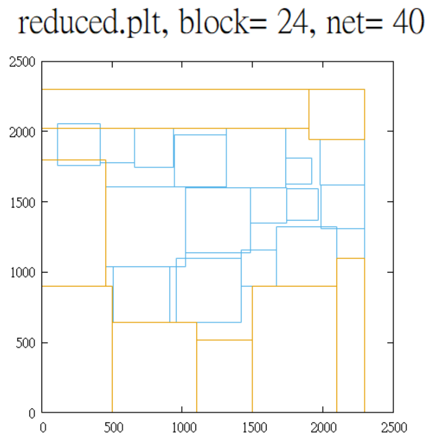
- The resultant force is calculated for each block, the directions of the resultant force is first tried when choosing where we want to move.
- Directions opposite of the resultant force is allowed, since we want to remove the overlapping area as much as possible.
- Blocks move one step at a time.

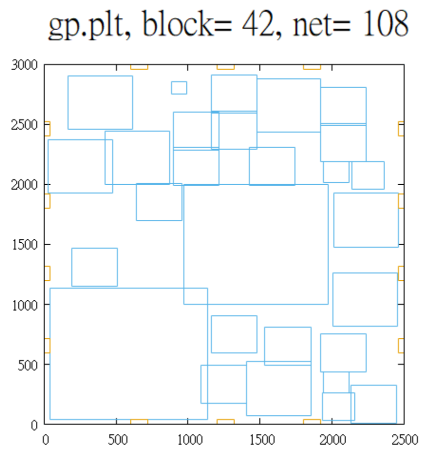
EXAMPLE

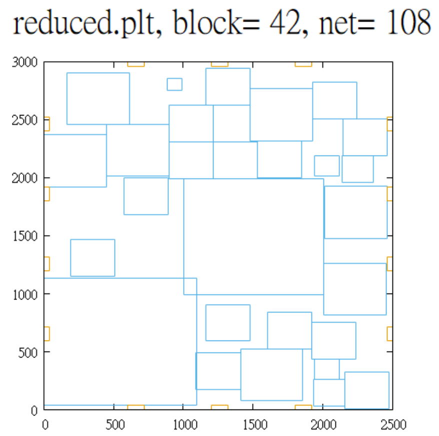












Global Placement overlapping area: 5.94%
Reduced overlapping area: 0.145%

Global Placement overlapping area: 7.76%
Reduced overlapping area: 0.751%

Global Placement overlapping area: 3.67%
Reduced overlapping area: 0.536%

BLOCK SPREADING

- Because other blocks may block the ones we want to move, we implemented a block spreading algorithm to increase the chance for other blocks to have space to move.

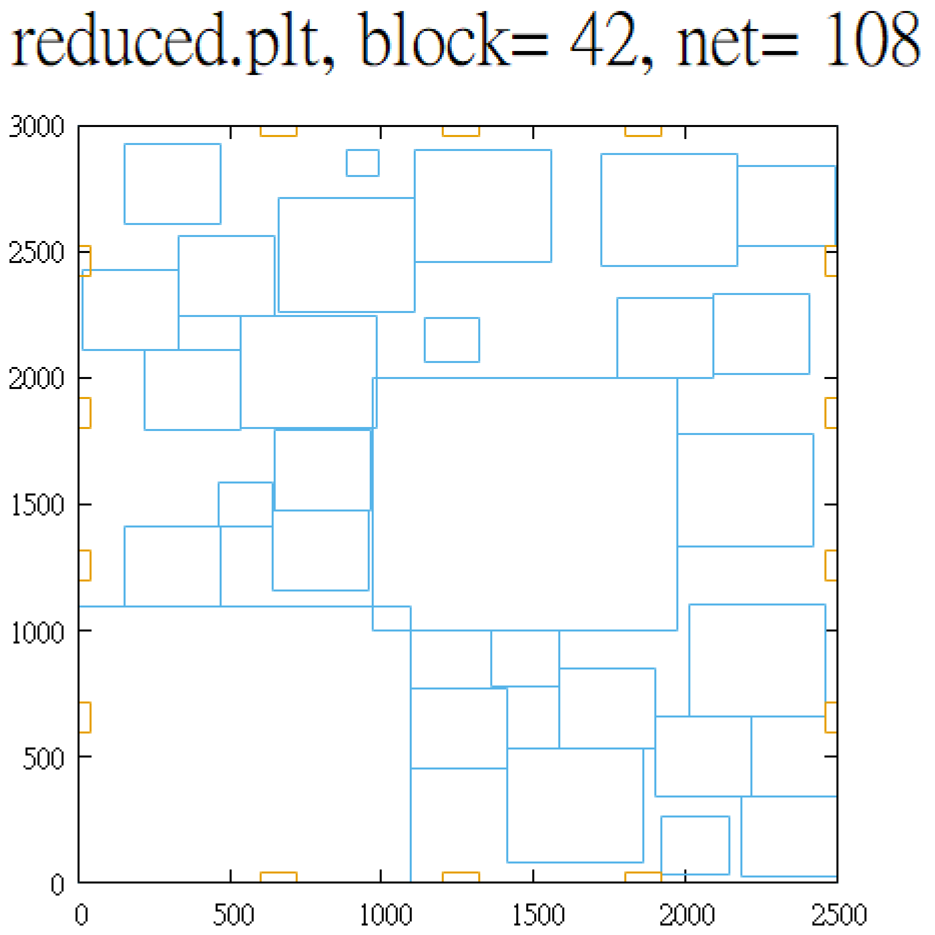
Flow:

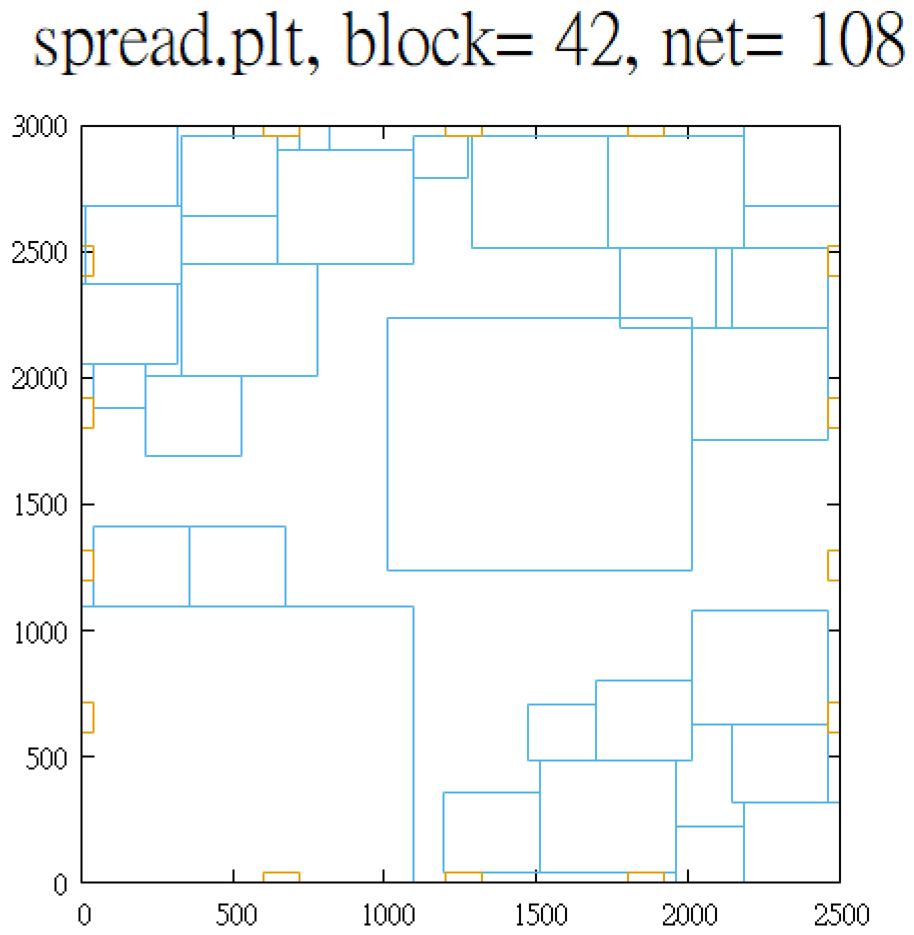
- First determine which direction should the block be moved, then use the same algorithm as the overlap reduction function for moving (but allow to an overlapping percent of 10% if that block already exist overlap).

How to determine where should the block move:

- First calculate which quadrant the block is in, then the possible direction of movement can only be ones pointing to the quadrant (ex: first quadrant is right and up).
- For blocks that aren't overlapped with others and blocks that only overlapped with soft block, the direction that won't overlap with new hard blocks and overlapped soft block's number remains the same is a legal direction.
- For blocks that overlapped with only hard blocks and both soft and hard blocks, as long as the number of overlapping blocks doesn't increase, and the already overlapping area <= 10%, then it is a legal move.

EXAMPLE





The large block can't move in the reduction phase since it can move toward any direction to reduce the overlapping area, but it can be further move after spreading.

LEGALIZE

- If the previous steps can't eliminate all overlapping blocks, then addition legalize is needed to remove all overlapping areas so that the placement is legal.

Flow (1st part):

- First determine if there exist a direction that if we increase size toward that direction the overlap area won't increase. If such direction exist, we start growing our block toward that direction until previous constraint isn't met (including the all the shape constraint), or the rectangle ratio is met.
- If the rectangle ratio is met, we start to shrink our block. The direction of shrinking is determined by which one can reduce the most amount of overlapping area. We shrink until the shape constraint is not met or the overlapping area doesn't decrease anymore.
- When shrinking is done, we check if the area is larger than the smallest legal area, if not, we start finding the direction to grow and grow until all constraint are met.
- If the rectangle ratio isn't met in the first time, we proceed to the second part.

What happen if shrinking and growing (resize) failed?

- If resize failed, we failed to legalize and the whole program is done again (to generate new placement).

CONT'D

- If the first part of legalization failed, we then proceed to the 2nd part. In the second part, we allow blocks to eat up more hard blocks (within constraint), in order to more easily legalize the block.

Flow (2nd part):

- First determine if there exist a direction that if we increase size toward that direction the hard module overlapped number doesn't increase, in other words, we allow block to grow and eat up more hard blocks (<18%).
- We start growing in those direction until the previous mentioned constraints are not met, or the rectangle ratio is met.
- If the rectangle ratio is met, we do the same thing (resize) as previous part until all the constraint are met, or we fail the legalization and we redo the whole flow.
- Note that (same as the first part), if there exist multiple direction we can start growing and one of the direction failed to met the constraint at some point, it'll still grow toward other direction.

What happen if we can't grow toward any direction and the rectangle ratio isn't met?

- Legalization failed and the whole program is done again (to generate new placement).

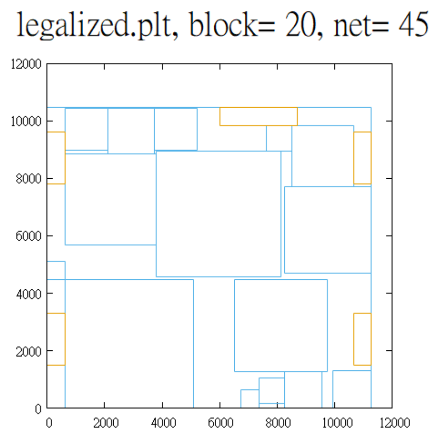
WIRELENGTH REDUCTION

- Because we spread the blocks for easier legalization, the wirelength will be longer than the initial Global placement most of the time, we reduce the wirelength for better performance.

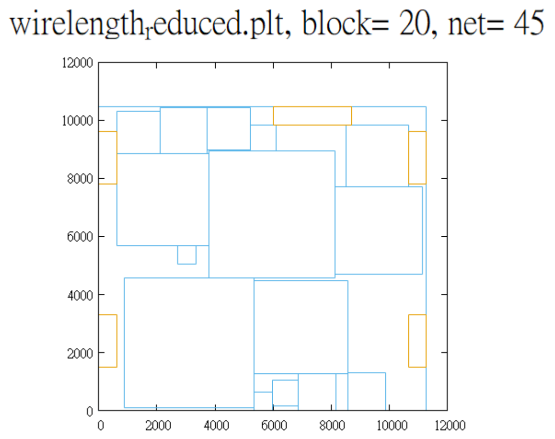
Flow:

- We sort the blocks with decreasing connectivity into a list (Multimap is used).
- Calculate the zero-force location for the blocks.
- Calculate the direction to move (toward zero-force location).
- Only move in the direction toward the zero-force location, until the overlapping area is greater than zero.
- We do the previous three steps for all the soft blocks.
- We loop until the nothing can move or over 1000 times.

Example:

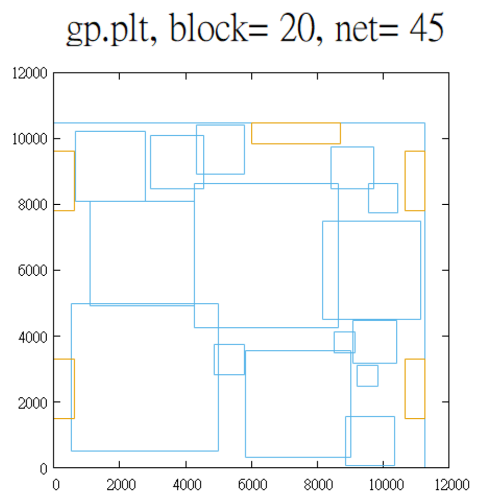


Legalized placement wirelength: 313554260

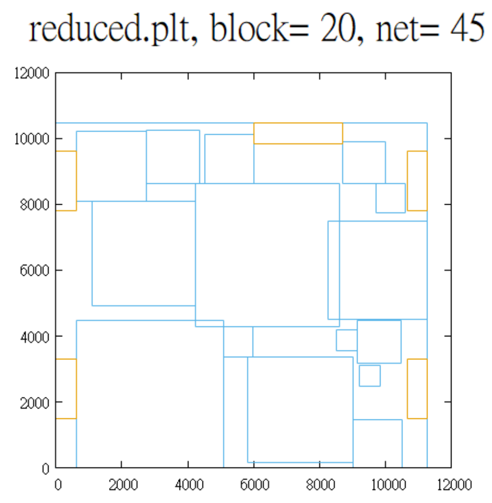


Legalized placement wirelength: 292620236

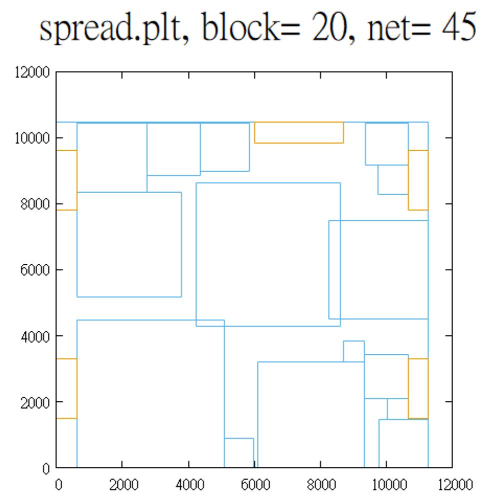
EXAMPLE (FULL FLOW)



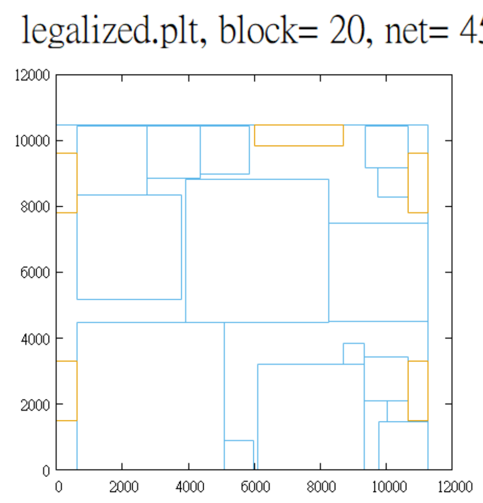
Overlap: 5.1724%



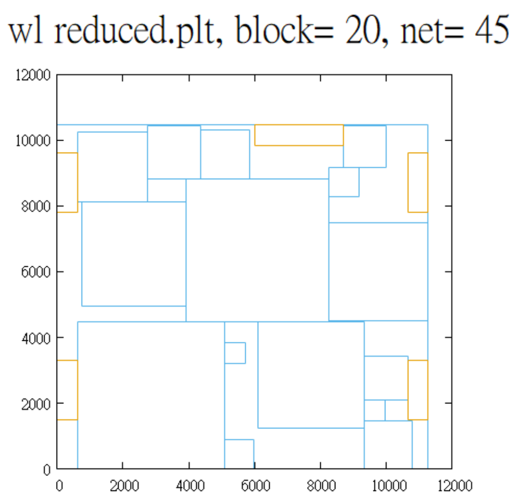
Overlap: 2.027%



Overlap: 2.027%

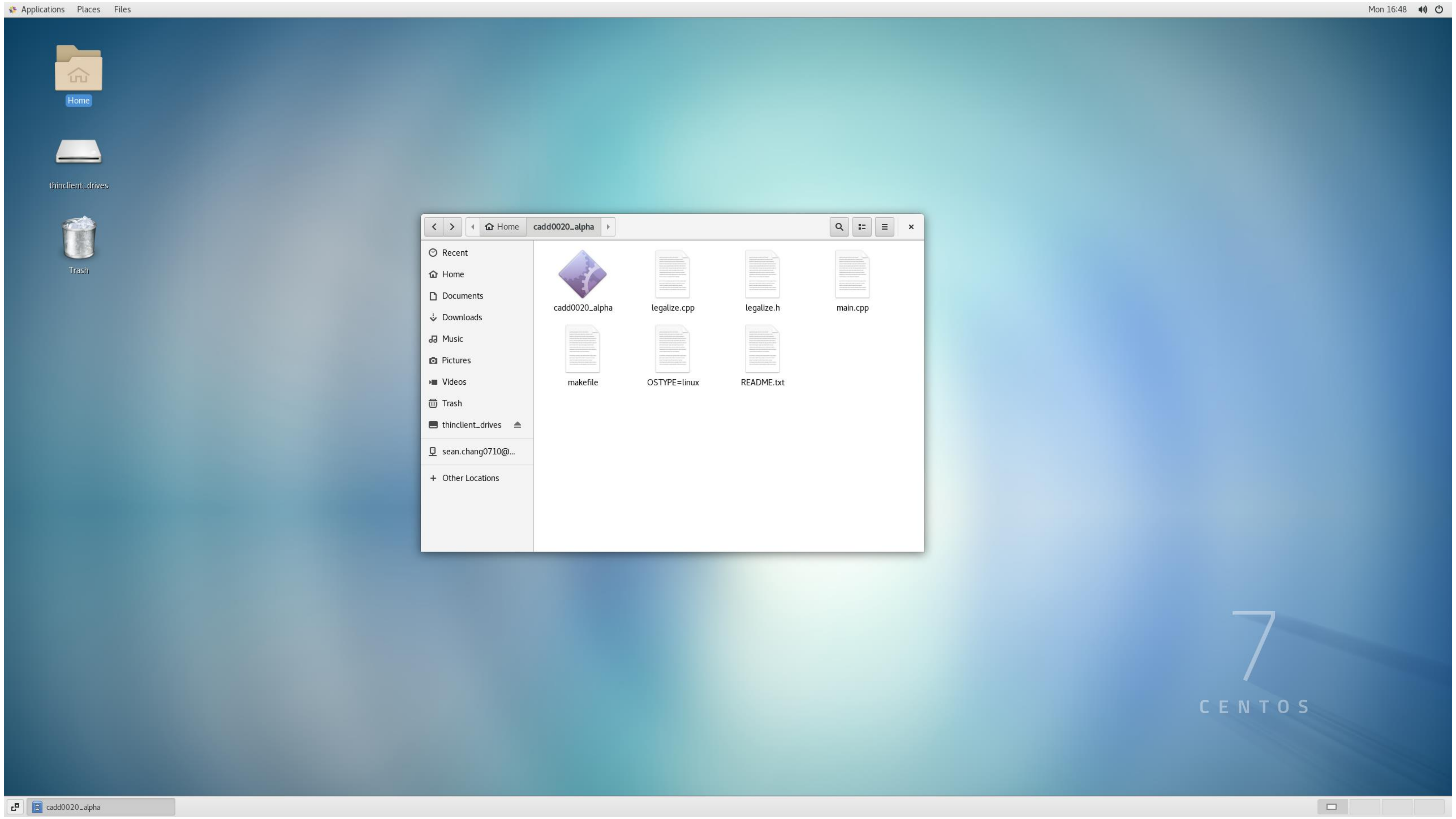


Overlap: 0%
Wirelength: 359985876



Overlap: 0%
Wirelength: 348964796

SUBMIT PROOF



THANK YOU