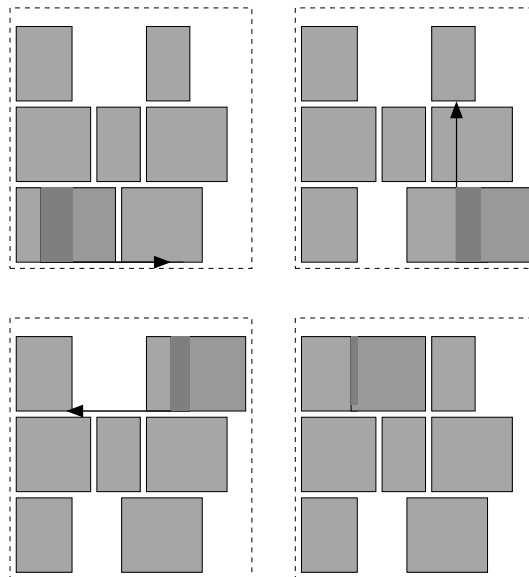


Optimization in VLSI Design.

Final Placement.



Jens Egeblad

September 30th, 2004

Overview

- Recapitulating
- Final Placement Overview
- Simulated Annealing approach
- Guided Local Search
- DOMINO
- Mongrel's Optimal Interleaving
- Summary

The Placement Problem (Recap)

Purpose: The objective of placement is to transform logic description into physical *minimal net-length* layout of *modules*.

The Problem: The placement problem is to place modules within chip-area such that the net-length is minimized.

The Placement Problem is **NP-hard!**

Net-lengths: While we solve The Placement Problem we *estimate* the net-length as either star, clique or bounding-box (So we avoid messing with complicated *RSMTs*).

Global/Final Placement: Solution procedures for solving the placement problem are divided into two groups: Global and Final Placement.

Global Placement deals with many modules and produces good solution within minutes and few hours.

Final Placement starts with a Global Placement solution and usually improves it by about 15 – 20% by moving one or few modules at a time. *Running times* are from hours to days.

Why do We Need a Final Placement Stage?

Global Placement heuristics do not produce optimal placements because:

- They often use quadratic net-lengths. Quadratic net-lengths do not reflect wire-length accurately. (although linearization schemes improve this).
- They deal with overlap by spreading modules somewhat arbitrarily.
- They optimize *many* modules at one time.

So the solution is not likely optimal. Therefore we need Final Placement.

However Global Placement heuristics are assumed to

- place modules with high connectivity close to each other,
- place modules in the same region as they would be in an optimal placement.

Then Final Placement heuristics are expected to find a *good* nearby local minimum from a global placement solution.

Combined the two heuristics may find the global optimal solution (But this is unlikely)!

Final Placement Overview

Final Placement heuristics often works as follows:

- Begin with a solution from Global Placement. Often without overlap.
- Iteratively improve location of one or several modules.
- Some heuristics allow a *little* amount of overlap while optimizing.

Why do some heuristics allow a little overlap?

Because this way only the modules which are moved affect the net-length which makes it a lot easier to *evaluate* a new solution.

Net-Model:

For *Global Placement* heuristics use The Quadratic Star or Clique net-models to optimize with. Why?

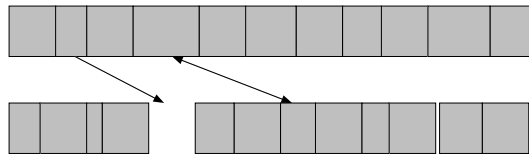
(Because unconstrained solutions spread the modules)

For *Final Placement* we use the Bounding-Box net-model. Why?

(Because bounding-box is a far more accurate estimate of RSMT).

Simple Local Search Neighborhood with SA

Select one module randomly and move it to a new location, or select two modules randomly and exchange them.



Calculate the change of net-length (We need only to consider connected nets).

If better, accept the new solution otherwise revert to old solution.

But: *What about overlap?* Overlap can be handled by extending the objective function with a *penalty term*:

$$\text{minimize } \sum_{n \in N} BB(n) + \lambda \cdot \sum_{m_i, m_j \in M} \text{area}(m_i \cap m_j)$$

Use Simulated Annealing (SA) to control acceptance randomly. Many “bad” moves accepted in the beginning, and few late in the process.

Apply a legalization step to remove overlap completely.

This is the foundation of the *successful* TimberWolf line of placers.

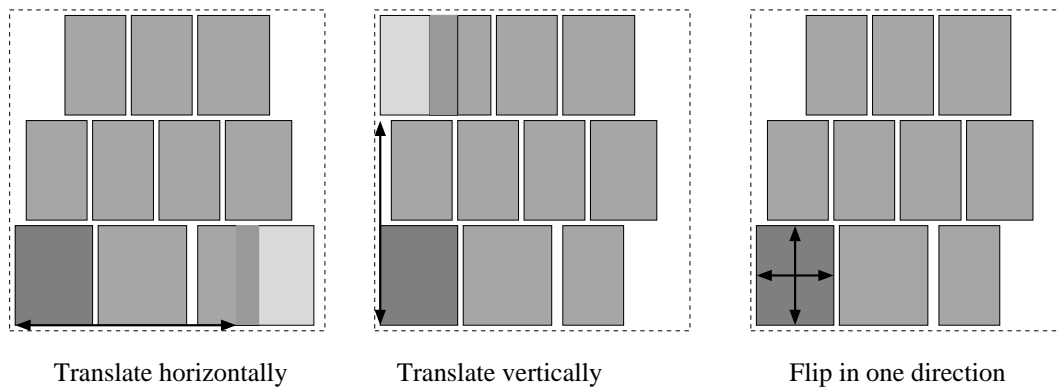
Guided Local Search (Local Search)

DIKU Research from 2000 by Oluf Færø, David Pisinger og Martin Zachariasen.

Objective function:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \beta \sum_{n \in N} BB(n) + \sum_{m_i, m_j \in M} area(m_i \cap m_j)$$

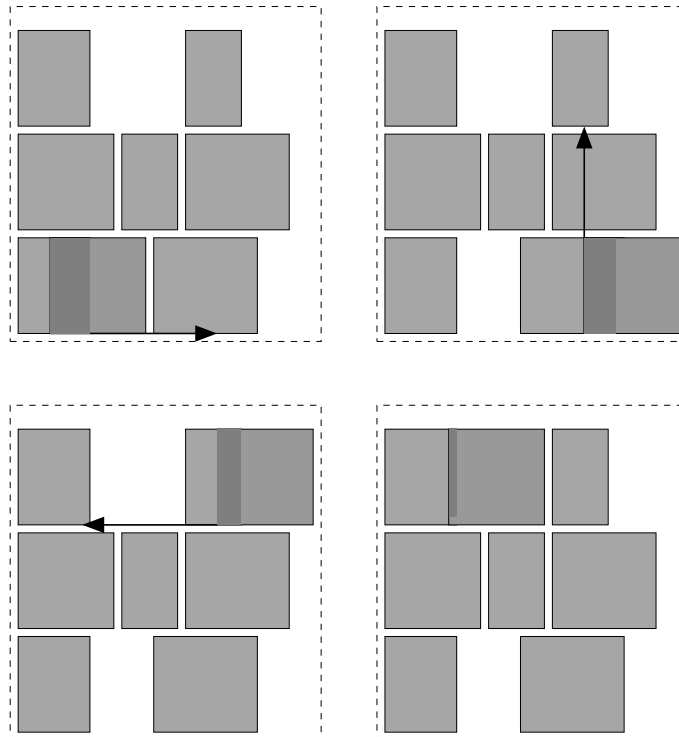
Local Search Neighborhood: Pick a module and *either*



Iteratively choose the move which will reduce $f(\mathbf{x})$ most.

Guided Local Search (2)

Example: (Without net-length)



...Local minimum.

Local minima have either of two properties:

- No *single* module can be moved because this would cause more overlap in the solution than reduction in net-length.
- No *single* module can be moved because this will not cause any reduction in net-length.

Applying GLS

Augmented Objective Function:

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda_1 \sum_{m_i, m_j \in M} p_{m_i, m_j}^o \cdot I_{m_i, m_j}^o + \lambda_2 \sum_{m_i, m_j \in M} p_{m_i, m_j}^c \cdot I_{m_i, m_j}^c,$$

where:

- $\lambda_1 > 0$ and $\lambda_2 > 0$ are used to fine-tune the heuristic.
- p_{m_i, m_j}^o is a *penalty term* of overlap between modules m_i and m_j .
- I_{m_i, m_j}^o is 1 if m_i and m_j overlap. 0 otherwise
- p_{m_i, m_j}^c is a *penalty term* of distance between *connected* modules m_i and m_j .
- I_{m_i, m_j}^c is 1 if *connected* modules m_i and m_j has a rectilinear distance ≤ 0 and 0 otherwise.

Penalty terms are initially 0. Whenever we reach a local minimum we increase the penalty terms of the pair of modules which overlap the most and the pair of connected modules which are farthest away.

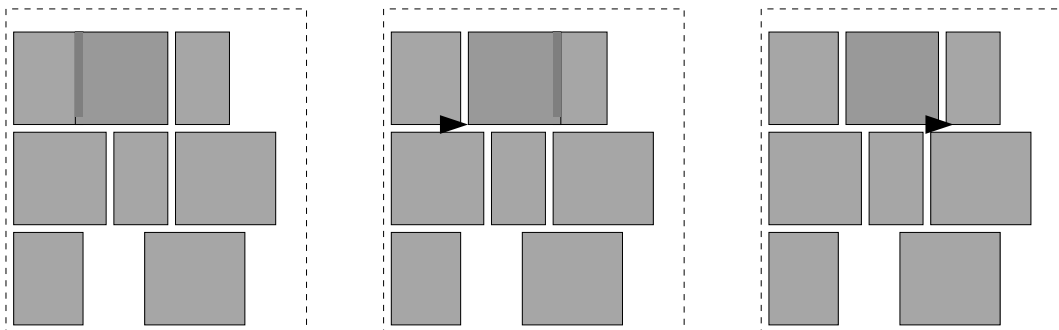
However to avoid penalizing the same pair of modules each time we divide by the current penalty term. E.g. $\mu = \frac{\text{area}(m_i \cap m_j)}{1 + p_{m_i, m_j}^o}$.

GLS Escaping Local Minima

Augmented Objective Function:

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda_1 \sum_{m_i, m_j \in M} p_{m_i, m_j}^o \cdot I_{m_i, m_j}^o + \lambda_2 \sum_{m_i, m_j \in M} p_{m_i, m_j}^c \cdot I_{m_i, m_j}^c,$$

Continuing our example:



Penalize

GLS Overview

Guided Local Search for Final Placement:

- Local search neighborhood consists of *translations* and *flipping* of *one* module.
- The move which *reduces* the objective function *most* is chosen.
- *Iteratively* applying this will put us in a local minima.
- Whenever we encounter a local minima *penalize*. The result: We can move into a new part of the solution space.

Neighborhood consists of all translations. Naively we would try each one and choose the best one.

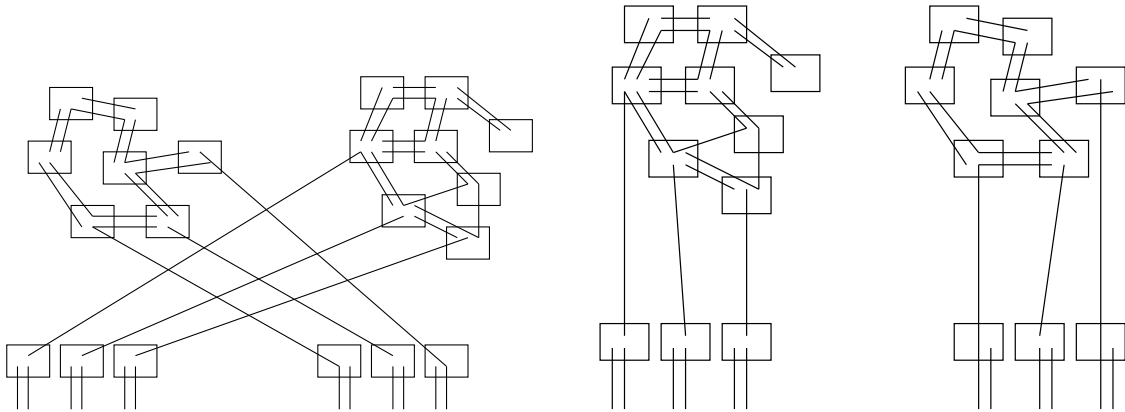
However, the neighborhood can be search in **polynomial time**. (The overlap part is an exercise for next week!)

GLS is extremely good (but a little slow) for VLSI-Problems.

This GLS approach has also applied with much success on **Bin-Packing Problems** and **Packing of Irregular Polygons**.

Optimizing Many Modules at a Time

Net-Length Local Minimum:



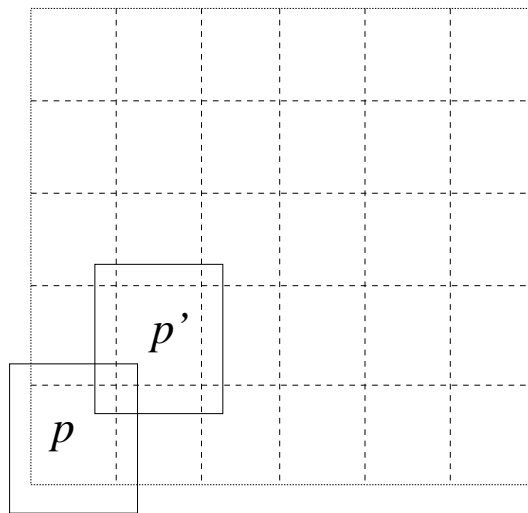
Solution: Move more modules at a time.

DOMINO (Overview)

DOMINO is from 1991 by the GORDIAN-team.

Optimizes Standard-Cell problems (in column form).

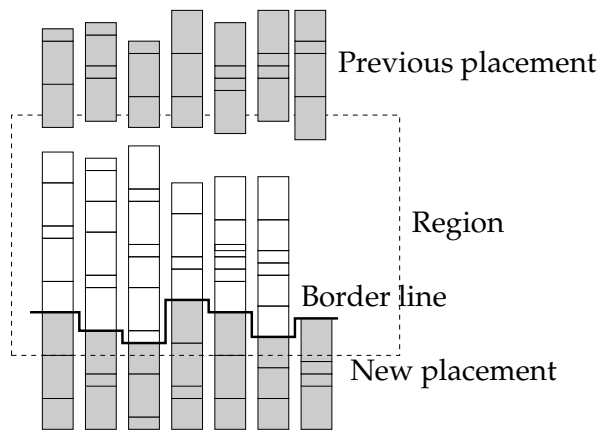
DOMINO works by considering one region a time.



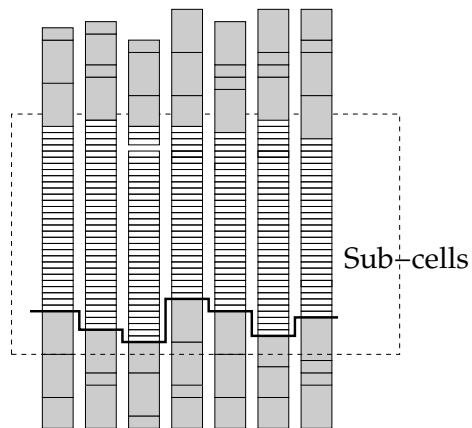
Traverse regions from bottom to top.

DOMINO (Sub-cells)

When a region is considered modules below that region are already placed.:



The columns are divided into "sub-cells"



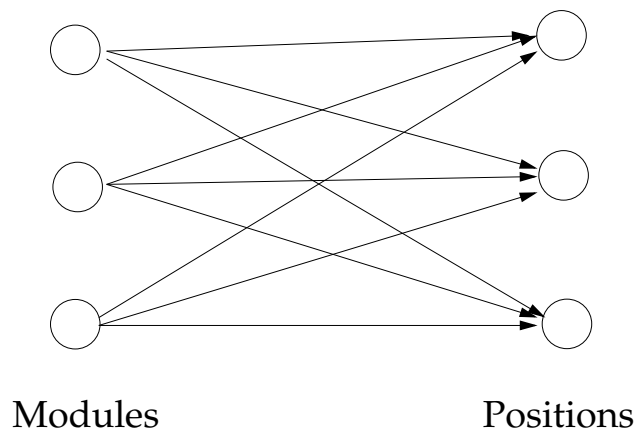
Each sub-cell corresponds to a position.

DOMINO (Flow Problem)

Each module m is divided into $u(m)$ units of sub-cell size.

Placing the modules can now be formulated as a flow problem:

- Each module m corresponds to a *source* of $u(m)$ units.
- Each sub-cell s position corresponds to a *sink* of 1 unit.
- We add an arch (m, s) between each source m and each sink s .
- Capacities $b_{m,s}$ on all arcs are 1.
- Edge costs $c_{m,s}$ are... Well, considered in a moment.



DOMINO (Edge Costs)

The cost of placing a *sub-cell* of module m at a location s should reflect the cost of placing *module* m at s .

Let $c_{m,s} = \sum_{n \in N(m)} L_{n,m,s}$, where $N(m)$ is the set of nets connected to m .

When a net n only connects m of all modules in the region p ,
Then:

$$L_{n,m,s} = [\text{The length of } n \text{ when } m \text{ is placed at } s]$$

But, when n connects more modules in p :

$$L_{n,m,s} = [\text{The length of } n \text{ when } m \text{ is placed at } s],$$

where the net-length is calculated *only* for modules *outside* p and m (Other modules in p are *ignored*).

DOMINO (Wrap-Up)

For each region DOMINO solves the flow problem.

It Places a module m in the column containing most of it's sub-cells and determine order in the column by center of gravity of sub-cells.

Regions are limited to 20–40 modules. To keep the problems simple.

Improve iteratively – Optimizing regions from bottom to top, bottom to top, bottom to top, bottom to top, bottom to top,...

DOMINO's Neighborhood:

The neighborhood is *large*. We consider many modules at a time.

May escape local minima since net-lengths are inaccurate. (Feature or bug?)

Mongrel's Final Placement (Overview)

Mongrel is a fairly new placement heuristic (2000).

It consists both of Global and Final Placement heuristics for Standard-Cell problems and uses a variety of *interesting* methods.

We will only consider the “optimal interleaving” part here!

The neighborhood exchanges k modules.

However, we cannot exchange k modules at k positions optimally in efficient time!

So the idea is to reduce this neighborhood.

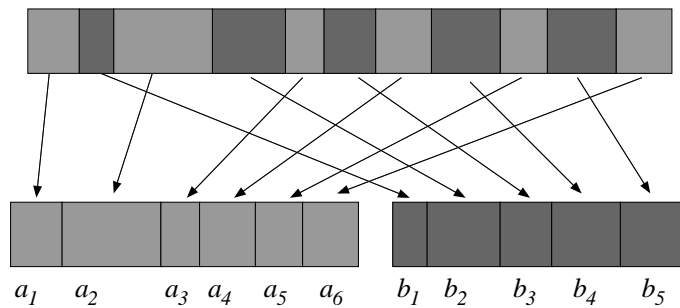
Mongrel's Final Placement (Interleaving)

Given two sequence of modules A and B we may produce an optimal "merge" or interleaving of the two sequences.

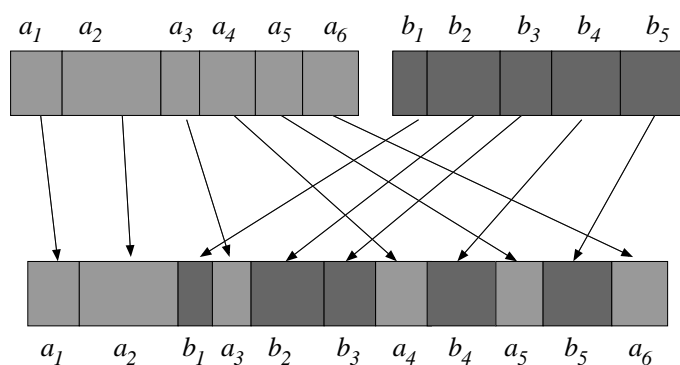
So begin by extracting part of a row of modules:



Divide it into two sequences A and B randomly.



Do an optimal merge of the two sequences. Modules in sequence A maintain their current order and modules in sequence B must maintain their current order.



Use the new sequence. It is optimal!

Mongrel's Final Placement (Dynamic Programming)

Given two sequences $A = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_m)$:

- let $S_{i,j}$ denote the *optimal interleaving* of $A_i = (a_1, \dots, a_i)$ and $B_j = (b_1, \dots, b_j)$.
- let $C(S_{i,j})$ be the cost of horizontal net-length within the window corresponding to $S_{i,j}$.
- Because $C(S_{i,j})$ is limited to the window, the optimality of $S_{i,j}$ is independent of any ordering of modules we add later on.

Therefore we can describe $S_{i,j}$ as a recurrence:

$$S_{0,0} = \emptyset$$

$$S_{i,j} = \begin{cases} S_{i-1,j}a_i & \text{if } C(S_{i-1,j}a_i) < C(S_{i,j-1})b_j \\ S_{i,j-1}b_j & \text{otherwise} \end{cases}$$

Which can be used for dynamic programming with a $m \times n$ table.

Running time: $O(mn + P(n + m))$, where P is the total number of pins on modules m and n

(Not quite as straightforward to see as the article claims!)

Final Placement Summary

Final Placements heuristics improve Global Placements solutions.

Final Placement heuristics almost always use the Bounding-Box Net-Model.

Some consider one module or two modules and allow overlap.

Others consider many modules to better escape local minima.

There are quite a bit of approaches often driven by meta-heuristics such as Genetic Algorithms, Tabu Search, Simulated Annealing (also without overlap). Etc.

But few approaches consider many cells in each iteration. (This may be the way!)

However, beware: Simple swaps alone may improve Global Placements by 10 – 15%. But e.g. *cyclic* moves with $k \geq 3$ modules does not reduce the length by more than a few percent.