swi 2002

pp. 77 – 90

CHAPTER 5

Magma Design Automation: Component placement on chips; the "holey cheese" problem

Rachel Brouwer, Thijs Brouwer, Cor Hurkens, Martijn van Manen, Carolynne Montijn, Jan Schreuder, JF Williams.

Abstract. The costs of the fabrication of a chip is partly determined by the wire length needed by the transistors to respect the wiring scheme. The transistors have to be placed without overlap into a prescribed configuration of blockades, i.e. parts of the chip that are beforehand excluded from positioning by for example some other functional component, and holes, i.e. the remaining free area on the chip. A method to minimize the wire length when the free area is a simply connected domain has already been implemented by Magma, but the placement problem becomes much more complex when the free area is not a simply connected domain anymore, forming a "holey cheese".
One of the approaches of the problem in this case is to first cluster the transistors into so-called macro's in such a way that closely interconnected transistors stay together, and that the macro's can be fit into the holes.
One way to carry out the clustering is to use a graph clustering algorithm, the so-called Markov Cluster algorithm. Another way is to combine the placement method of Magma on a rectangular area of the same size as the total size of the holes, and a min cut-max flow algorithm to divide that rectangle into more or less rectangular macro's in such a way that as little wires as possible are cut.
It is now possible to formulate the Quadratic Assignment Problem that remains after clustering the original problem to one with 100 up to 1000 macros. There exists a lot of literature on finding the global minimum of the costs, but nowadays computational possibilities are still too restrictive to find an optimal solution within a reasonable amount of time and computational memory. However, we believe it is possible to find a solution that leads to a acceptable local minimum of the costs.

1. Introduction

One of the steps in the design process of chips is the positioning of every single transistor or "cell" on the chip. This means that, given a certain wiring scheme, i.e. the scheme describing the connections between the cells, and taking into account the - relatively few - cells with a prescribed position, the positioning of the various cells has to be determined while under the following conditions. First, the cell must be placed within a certain rectangle, the so-called core area; next, the cells are not allowed to overlap; and finally, the

total wire length must be minimized, as the cost of a chip is proportional to the total wire length. For this problem, many algorithms are known, each one with its specific pros and cons.
The problem becomes more difficult when large parts of the core area are excluded from positioning, often due to large, functional components that were placed beforehand (e.g. memory, or components designed by other companies), creating so-called blockades. The remaining "free area" within the core area is usually comparable to a "holey cheese". Obviously, the cells cannot be placed on the blockades, and this additional requirement makes the positioning problem significantly harder. Figure 1 shows an example of a typical "holey cheese". The filled areas are allowed and form the so-called "holes", the white ones are blockades. Notice that the free area is strongly disconnected.

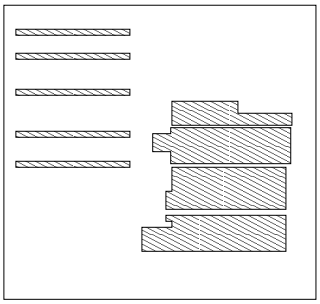

FIGURE 1. A typical example of a "holey cheese" configuration: the transistors may be placed in the filled zones, the white remaining areas are blockades.

The current algorithms of Magma suffice for simply connected domains. However, in "holey cheese" cases they often end up in local minima for the costs that are far from optimal. The purpose of our group during the study group week was to find an algorithm enabling Magma to find more optimal placements of the cells in the case of a "holey cheese".

To this end we decided to make the following approach of the problem: first regroup the strongly connected cells in more or less equally sized "macros", then place these macros in the holes in such a way that the wire length is minimized. We present two possible approaches for the regrouping of the cells. The first one, dealt with in section 2.1, departs from the wiring scheme and uses a clustering algorithm. The second one, subject of section 2.2, is a combination of a preprocessing step using the original Magma software, followed by a repeated application of a min cut-max flow algorithm. Finally,

in section 3, we discuss the method to minimize the wire length, formulating a Quadratic Assignment Problem (QAP).

2. Clustering the cells

2.1. The Markov Cluster Algorithm. We believe that the Markov Cluster Algorithm (MCL) provides a good method to group the cells in macros. This algorithm uses the notion of random walk for the retrieval of cluster structure in a graph. In a random walk at each cell the direction to be followed is given by chance. Imagine a vast collection of random walks, all starting from the same cell. Walkers will in general follow different paths. An observer floating high above them will see a flow: the crowd slowly swirles and disperses, much as if a drop of ink is spilled into a water-filled tray.
The aim of a cluster method is to dissect a graph into regions with many interconnections inside, and with only a few interconnections between regions. Once inside such a region, a random walker has little chance to get out. The idea behind MCL is very simple. Simulate many random walks (or flow) within the whole graph, and strengthen flow where it is already strong, and weaken it where it is weak. By repeating the process an underlying cluster structure will gradually become visible. The process ends up with a number of regions with strong internal flow (clusters), separated by 'dry' boundaries with hardly any flow.
We refer to the PhD-thesis of Stijn Van Dongen[6] for a detailed review of this algorithm.

2.2. Constructing macros with the min cut-max flow algorithm. The method of constructing macros with a min cut-max flow algorithm departs from a square S with surface A_s over which all the cells have been positioned in such a way that they do not overlap and that their connectivity has already been taken into account, meaning that the highly interconnected cells are already put together. This positioning of the cells within a square is the result of a preprocessing method implemented by Magma.
Schematically the procedure is as follows: put a grid over square S of which the grid lines may slightly be deformed. Then use a min cut-max flow algorithm to deform the grid lines in such a way that they cut as little connections as possible. The cells contained in the resulting grid cells then form the macros.

2.2.1. *Restrictions on the macros.* Assume the number of macros we want to make is n, and let A_h and A_m be the total surface of the holes and the average size of the macros, respectively. There are some restrictions on the number and size of the macros. First, as will be shown in section 3, the computationnal hardness of the QAP imposes the number of macros to be no larger then 1000. Also creating a large number of macros might result in breaking up highly connected parts, which looks inefficient since

it is probable that the QAP routine will put them together again. But the number of macros shouldn't be too small either since then the QAP might have no effect. Second, we want the average macro to fit at least twice in the smallest hole; this constraint is not very strict when $A_s \ll A_h$ since we then may choose to ignore very small holes, however, if $A_s \approx A_h$, it is necessary to use all possible space.

2.2.2. *Defining the adjustable grid.* Figure 2 shows how the adjustable grid can be defined: each grid cell should contain exactly one striped rectangle and the grid lines (dash-dotted lines) can be moved freely within the white areas.

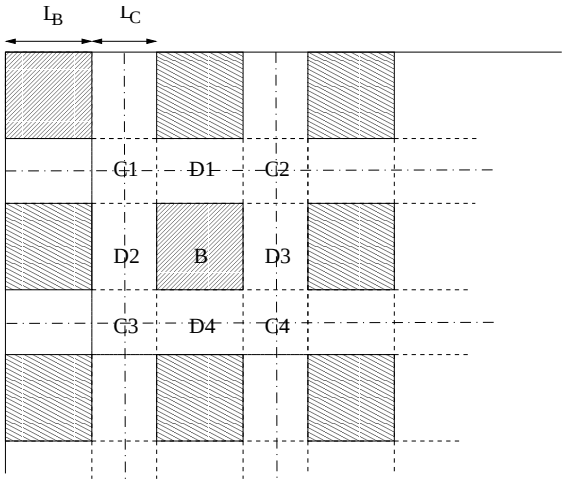

FIGURE 2. The definition of the adjustable grid: each grid cell should contain exactly one striped rectangle and the grid lines (dash-dotted lines) can be moved freely within the white areas.

Define squares B_{ij} (the striped ones in figure 2) with edges of length L_B, the upper left and right corner points given by the points $\{(i-1)(L_B+L_C), (j-1)(L_B+L_C)\}$ and $\{(i-1)(L_B+L_C), (j-1)(L_B+L_C)+L_B\}$, respectively, and the lower left and right corner points being $\{(i-1)(L_B+L_C)+L_B, (j-1)(L_B+L_C)\}$ and $\{(i-1)(L_B+L_C)+L_B, (j-1)(L_B+L_C)+L_B\}$, respectively, for $i = 1...\lceil\sqrt{n}\rceil$, for $j = 1...\lceil\sqrt{n}\rceil$. Each square B_{ij} is now separated of a neighboring square by a distance L_C. We assign all the cells contained in B_{ij} to macro A_{ij}, and we still have to assign the cells contained in the surrounding areas to one of the neighboring macros. This we will be done with help of a min cut-max flow algorithm.

For the lengths L_B and L_C we suggest:

$$L_B = (A_s/2n)^{1/2},$$

$$L_C = (\sqrt{2} - 1)L_B.$$

That is, exactly half of the surface is assigned to the macros beforehand. With $L_C = (\sqrt{2}-1)L_B$ we obtain $n \cdot (L_B + L_C)^2 = A_s$. It is of course possible to adjust these numbers. If we take L_C bigger and L_B smaller, we have less surface assigned beforehand, hence it seems reasonable to assume that we will obtain a better assignment. On the other hand, this has also disadvantages : the size of a macro will vary more, and hence we may encounter problems with the placing of these macros in the holes if A_s is close to A_h. Moreover, the speed of the min cut-max flow algorithm depends on the size of the graph. However, the algorithm can be executed in polynomial time, so this does not have to lead to considerable delay.

2.2.3. *The assigning procedure with a min cut-max flow method.* For all cells c, we make a list of possible assignments to the macros. Notice, from figure 2 that the cells in the areas $C_1, ...C_4$ belong to the surrounding areas of four squares, whereas the cells in the areas $D_1...D_4$ belong to the surrounding area of only two squares. Notice also that connections that reach over the borders of a square B_{ij} and its surrounding area will automatically be cut. We then assign all cells c with their midpoints inside B_{ij} to macro M_{ij}. For each cells c in the surrounding area of B_{ij} we make a list with all the possible macros it can be assigned to.
We now construct a graph consisting of all cells in B_{ij} itself and in its surrounding area. We contract all cells inside the square B_{ij} to one point S, and we contract all cells outside the surrounding area to one point T, while keeping their connecting wires as they are (See Figure 3.) We now apply a min cut-max flow algorithm to determine a minimal cut set and assign cells to M_{ij} according to this cut set. The min cut-max flow theorem and algorithm have first been investigated in 1956 by Ford and Fulkerson. All textbooks on graphs and flows contain the theorem and often also an algorithm. For some recent versions of the theorem and the algorithm we refer to Diestel[5], Gross[6] and Jungnickel[7]. For a cell that is not assigned to M_{ij}, we remove this macro from its list as a possible assignment. Notice that it is also possible that we have cells in the surrounding area that are not connected to either S or T. For these (small clusters of) cells we can choose an arbitrary macro. In our opinion, it is advisable to assign these cells in such way that it will lead to macros that do not vary much in size.

The min cut-max flow algorithm assures that for the graphs as we have constructed above, a minimum of connections will be cut. This does not take into account the length of these connections. It does keep clusters of highly connected cells in one macro. Note that "long" connections, that reach over two or more squares B_{ij} will be cut automatically. These connections will probably belong to nets that reach over big distances and that therefore

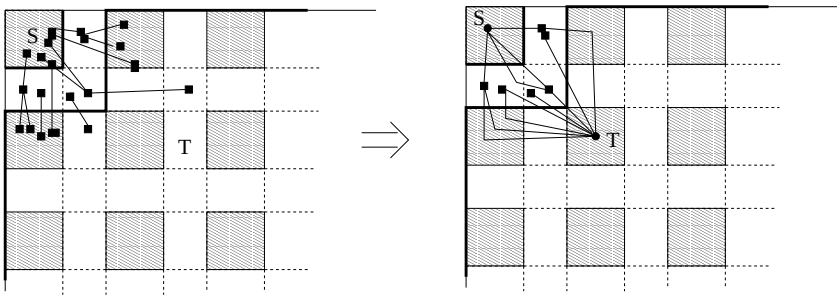

FIGURE 3. Sketch of the graph used in the min cut-max flow algorithm. The last figure shows the cluster (filled) obtained with the algorithm

must always be cut, no matter what way we choose to construct the macros. Note that the procedure above depends heavily on the ability of the Magma-procedure to keeping highly connected clusters together.

3. Formulating a new macro placement problem

After the preprocessing stage, the set of all movable cells c has been partitioned into n macros. For each movable cell c, let $M(c)$ denote the macro to which it belongs. Each macro μ has an area requirement $A(\mu)$, which is equal to the total area of the transistors in the macro: $A(\mu) = \sum_{c \in \mu} A(c)$. Whenever two macros contain transistors that appear in the same net, these macros are connected. Similarly, when a macro contains a transistor which is connected by a net with a fixed pin f, the macro is also considered to be connected to pin f. We elaborate on the connectivity structure between macros and fixed pins in the following section.

Besides the macros we are given m holes in the placement area, with total area not less than the total required transistor area. We consider the problem of assigning the macros to the holes in such a way that the expected resulting wire length — after refined placement of the transistors within each hole — will be as low as possible.

In order to properly define the problem, we first have to define the exact connectivity requirements, and make up our mind how to assign macros to holes in such a way that the resulting wire length between the macros is accounted for as precisely as possible.

3.1. Connectivity between macros. The connectivity between transistors and fixed pins has originally been defined in terms of *nets*, where a net is simply a subset of the collection of placeable and fixed pins. It is evident that after placeable transistors have been clustered into macros these connectivity requirements carry over.

Let N be an original net, that is, $N = \{c_1, c_2, \ldots, c_k\} \cup \{f_1, \ldots, f_l\}$ containing k transistors and l fixed pins, $k \geq 0, l \geq 0$. These pins have to be connected by some wiring network. This implies that this wiring network covers macros $M(c_1), \ldots, M(c_k)$ and fixed pins $f_1, \ldots, f_l$. See Figure 4 for an example.

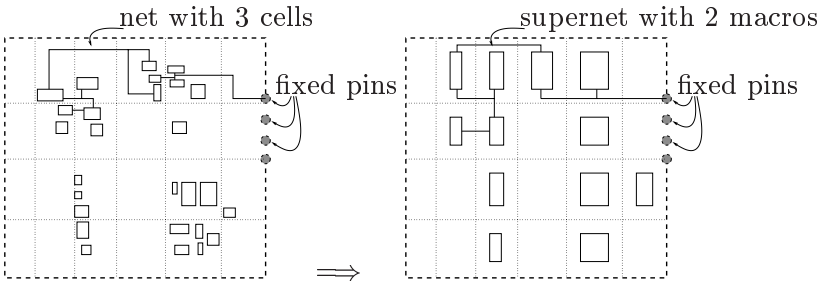

FIGURE 4. Partitioning into macros, connected by supernets

Note that in the macro formulation some of the requirements will be lost, in particular when $l = 0$. It may be the case that $M(c_1) = \ldots = M(c_k)$. In this case we have no information about the resulting wiring length for net N, other than that it will not be big, since it will not be stretching over two or more holes.

What matters is, how many **distinct** macros are covered by each supernet. Removing duplicates from $M(c_1), \ldots, M(c_k)$ we say that the *net* N induces a *supernet* $N' = \{M(c_1), M(c_2), \ldots, M(c_k)\} \cup \{f_1, \ldots, f_l\}$.

It is of interest to consider the numbers of nets and the connectivity of transistors, and to see how this carries over to macros. To play around with the problem we were given several instances of the Magma-problem, and for one of these we were actually given a layout of the transistors with a relative low wiring length, not taking into account that transistors can be placed only inside the prescribed holes.

The toy problem contains 310 fixed pins, and 2099 movable cells, and the wiring structure consists of 2234 nets. The total number of pin-net combinations is 8614. The net size varies from 2 to 288 pins, with an average of 3.58 pins per net. There are 1359 nets of size two, and 318 nets of size three. Each fixed pin was contained in a single net, and each movable cell was contained in between 2 and 7 nets, with an average of 3.96 nets per movable cell.

From the given layout in which the transistors were laid out more or less uniformly over a square we constructed a partition into 100 macros by subdividing this area in a 10 by 10 grid. Now the number of macro-supernet combinations was 4505, where one should note that as many as 1173 supernets cover only one macro, so there are 1061 supernets that cover

two or more macros, with an average of 3.15 macros per supernet. Among these 1061 supernets, 772 cover only two macros. The maximum number of macros covered by a supernet is 78. So, on average, each macro is contained in 45 supernets, of which 11.7 are singleton supernets. The number of macro pairs that have at least one supernet of size less than 10 in common, is 651. Hence, on average, each macro is connected to 13 other macros.

3.2. Subdividing holes into smaller areas. In order to get a better estimate of the ultimate wire length it seems appropriate to subdivide the holes into areas that are more or less equally sized, and such that the wire length within a hole can be neglected without introducing a too large error. The idea is to subdivide the target holes into such smaller areas, taking the midpoint of each area as the virtual placement position. The choice for M, the number of sub-holes, should depend on m, the number of original holes, as well as on the sizes of the holes, and on the number of macros n. M should preferably divide n, and the area of each sub-hole should be an integral multiple of the average macro area. Since the wire length estimate is more precise when more target holes are taken, we have chosen to take $M = n$, for the toy problem. See Figure 5 for an example.

A problem that arises is to split the target holes into the required number of approximately equally sized sub-holes. First one determines the average sub-hole area by $\hat{A} = \frac{1}{M}\sum_{h=1}^{m} \text{area}(H_h)$. Then hole H_h initially gets assigned $\lfloor \text{area}(H_h)/\hat{A} \rfloor$ sub-holes. Next the remaining $M - \sum_h \lfloor \text{area}(H_h)/\hat{A} \rfloor$ sub-holes are "evenly" distributed over the holes, in the same way as rest votes after an election have to be distributed. Once it is decided that hole H_h is subdivided into M_h sub-holes, the question is where to put these sub-holes so as get an even distribution. This depends on the aspect ratio (ratio of longer side over shorter side) as well as the number. For instance, it is not so obvious to subdivide an area of 20 by 40 into 5 more or less equal areas. The average area within the hole is $800/5 = 160$. One can take five rectangles of 8 by 20, or one of 8 by 20, and four of 16 by 10, or two of 8 by 20 and three of 12 by $\frac{40}{3}$.

Once it is decided how the sub-holes are defined we take the midpoints of these sub-holes as our target positions. Let (X_p, Y_p), denote the midpoint of sub-hole p, for $p = 1, \ldots, M$.

3.3. Assignment cost. Next we will try to assign macros to positions in such a way that each macro is assigned one position, and each position is assigned only one macro. When the clusters are more or less equally sized, and when there is enough slack area in the system such an assignment yields a more or less feasible layout. The cost of such an assignment should reflect the final wire length incurred. Now the actual wiring is done later so we can only estimate it. MAGMA is faced with the same problem and has chosen to

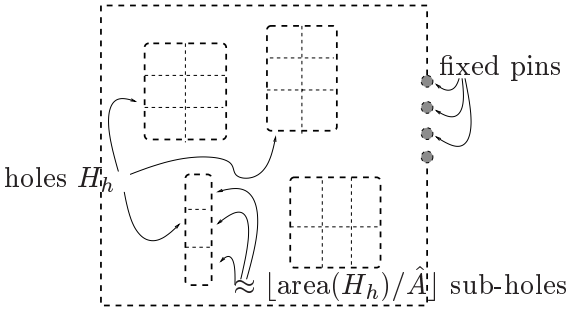

FIGURE 5. Subdivision of holes into subholes

estimate the wire length for a net as (half of) the perimeter of the bounding box of the pins inside a net. Adopting the same approach we could take as the wire length for each *supernet* half the perimeter of the bounding box of the fixed pins of the supernet and the midpoints of the sub-holes covered by the supernet.

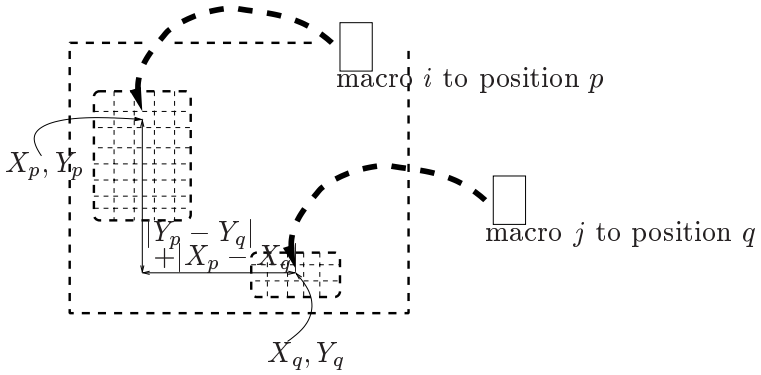

FIGURE 6. Assignment of macros to subholes

Again, this is cumbersome in the sense that such a cost function can only be evaluated once the total assignment has been made. We prefer to formulate a cost function that is of a more local nature. Now recall that in the toy problem many of the supernets only cover two or three macros. For a net of two pins at positions P_1 and P_2, the Manhattan distance $d(P_1, P_2)$ equals the (shortest possible) wire length and equals half the perimeter of the bounding box. For a net of three pins placed at positions P_1, P_2, and P_3, the perimeter of the bounding box is exactly equal to $d(P_1, P_2) + d(P_2, P_3) + d(P_3, P_1)$. Hence, in the estimate of the wire length each distance $d(P_i, P_j)$ contributes with a factor 0.5. For (super)nets of more than three pins it is impossible to tell a priori how much the distance $d(P_i, P_j)$ will contribute

to the perimeter of the bounding box. One may estimate the contribution of the distance between positions P_i and P_j when covered by a net of size $S > 3$ by something like $\frac{2}{S}d(P_i, P_j)$ or by $\frac{3}{S(S-1)}d(P_i, P_j)$. The latter one yields a true lower bound. Another choice could be to neglect wire length with regard to large supernets.

3.4. Formulation. Let $D_{pq} := |X_p - X_q| + |Y_p - Y_q|$ denote the Manhattan distance between positions p and q. Let macros i and j have connectivity C_{ij}, defined by $C_{ij} := \sum_\nu \gamma_\nu$, where the sum is taken over all supernets ν that cover both macro i and j, and where $\gamma_\nu = 1, 0.5$ or $\frac{3}{S(S-1)}$, if supernet ν has size $2, 3$ or $S > 3$, respectively. Note that the size of a supernet is the number of distinct macros and fixed pins that it covers. Then the contribution of the connectivity between macros i and j, when placed at positions p and q respectively, to the estimated wire length will be $C_{ij}D_{pq}$. Let $E_{ip} := \sum_f \gamma_{\nu(f)} d(P_p, f)$ denote the fixed cost associated with assigning macro i to position p. Here the sum is taken over all fixed pins f connected to macro i by a net $\nu(f)$.

We now set the problem in variables x_{ip} with x_{ip} equal to 1, if macro i is placed at position p, and 0, otherwise. The final mathematical problem to 'solve' is the following quadratic assignment problem

$$
\begin{array}{lll}
 & \text{Minimize} & \sum_{ip} \sum_{jq, j>i} C_{ij} D_{pq} x_{ip} x_{jq} + \sum_{ip} E_{ip} x_{ip} \\
\text{(QAP)} & \text{subject to} & \\
(1) & \sum_p x_{ip} = 1 & \forall i \\
 & \sum_i x_{ip} = 1 & \forall p \\
 & x_{ip} \in \{0,1\} & \forall i, p
\end{array}
$$

4. Attempting to solve the QAP

As the title of this section suggests, it is possible to formulate the global placement problem in an appropriate model, but it is not that easy to actually solve this problem to optimality. From literature [1], [2], [3], [4] we have found that even problems of moderate size (with $n = M = 30$) are very notorious. Quite recently a paper [2] has been published announcing the optimal solution of `nugent30`, by years of CPU, using a distributed computing network. This is a quadratic assignment problem with a background similar to that of the MAGMA problem.

One should keep in mind that it is not necessary to solve our problem to optimality, since it is already an approximation of an approximation. We have investigated methods to find a proper lower bound on the QAP-value. The problem is that such a lower bound is found by relaxing the original problem to something that is solvable. The actual solution of the relaxed problem may be far away from a decent solution of the original problem, as

we will see. However, even the *value* of the relaxed problem can be of use, since, if it is a proper lower bound, it may indicate the quality of any solution obtained by whatever way. For instance, it is possible to find assignments of macros to positions, by means of local search: simply start with any solution, apply small changes by exchanging the positions of two or more macros, and proceed until no such change leads to improvement. We have not implemented this approach but find it a true possibility.

Next we will describe some of the insights we found in the literature, and show how some of the easiest lower bounds can be effectively computed. We will give the reference and provide MAGMA with copies thereof.

4.1. Lower bounds. Each method to solve the QAP to optimality needs, at some point in time, a way of proving that the achieved result is best possible. In order to compute a lower bound for the general quadratic assignment problem of the form

$$
\begin{array}{rll}
 & \text{Minimize} & \sum_{ip}\sum_{jq,j\neq i} Q_{ijpq}x_{ip}x_{jq} \\
(\text{GenQAP}) & \text{subject to} & \\
 & \sum_p x_{ip} = 1 & \forall i \\
 & \sum_i x_{ip} = 1 & \forall p \\
 & x_{ip} \in \{0,1\} & \forall i,p
\end{array} \tag{2}
$$

one can rewrite the objective to $\sum_{ip} x_{ip}\sum_{jq,j\neq i} Q_{ijpq}x_{jq}$ and replace the last part by the solution of

$$
\begin{array}{rll}
 & \text{Minimize} & \sum_{jq,j\neq i} Q_{ijpq}x_{jq} \\
(\text{LAP})_{ip} & \text{subject to} & \\
 & \sum_q x_{jq} = 1 & \forall j \\
 & \sum_j x_{jq} = 1 & \forall q \\
 & x_{ip} = 1 & \\
 & x_{jq} \in \{0,1\} & \forall j,q
\end{array} \tag{3}
$$

To finally compute the lower bound, in literature known as the Gilmore-Lawler Bound (GLB), one has to solve one additional Linear Assignment Problem:

$$
\begin{array}{rll}
 & \text{Minimize} & \sum_{ip}(\text{LAP})_{ip}x_{ip} \\
(\text{GLB}) & \text{subject to} & \\
 & \sum_p x_{ip} = 1 & \forall i \\
 & \sum_i x_{ip} = 1 & \forall p \\
 & x_{ip} \in \{0,1\} & \forall i,p
\end{array} \tag{4}
$$

All these linear assignment problems can be solved efficiently using standard network algorithms, even for $n = M = 100$, which yields a 10,000 by 10,000 assignment problem.

Other lower bounds can be computed based on eigenvalue decomposition and projection methods. The main problem with these methods is that the matrix Q is not positive semi-definite. This means that the values one gets by relaxing the constraints $x_{ip} \in \{0, 1\}$ are very low, even negative.

4.2. The QAP in Koopmans-Beckmann form. Note that the cost coefficients have a special form, as the main part of the cost is the product of connectivity and distance. This special form allows for a fast computation of GLB. That is, in order to solve GLB one first has to compute the value $(\mathrm{LAP})_{ip}$, for each ip. When the Koopmans-Beckmann form applies, solution of $(\mathrm{LAP})_{ip}$ amounts to sorting the values C_{ij} (for fixed i, and for $j \neq i$) in non-increasing order, sorting values D_{pq} (for fixed p, for $q \neq p$) in non-decreasing order and computing the inner product of the two arrays. Let $< C_{i*}, D_{p*} >$ denote the value of this inner product. Then the Gilmore-Lawler lower bound for the MAGMA problem is given by

$$
\begin{array}{lll}
 & \text{Minimize} & \sum_{ip}(E_{ip} + \frac{1}{2} < C_{i*}, D_{p*} >)x_{ip} \\
(\mathrm{GLB-MAGMA}) & \text{subject to} & \\
 & \sum_p x_{ip} = 1 & \forall i \\
 & \sum_i x_{ip} = 1 & \forall p \\
 & x_{ip} \in \{0,1\} & \forall i, p
\end{array} \tag{5}
$$

4.3. Constructing a true solution. The GLB value is a true lower bound on the global location problem. The solution of GLB-MAGMA will actually give an assignment. However, this will probably only be good in the sense that the allocation of macros to 'bad positions' does select those macros that have a limited connectivity. But the assignment does not discriminate too much between macros with limited connectivity. So the main contribution of the GLB-solution is its value. Furthermore the GLB-solution could be used as the starting point for an exchange algorithm that can be set up in a local search frame work. This local search approach should be using the true quadratic cost function. By exchanging the assignment of a limited number of macros at a time, say two, three or four, one can effectively compute the change in the objective of a tentative exchange, and perform such an exchange as long as an improvement is made. It is mentioned in literature, that GLB gives a poor bound when the number n is high. In view of this observation, it may be wise to experiment with the number $n = M$, with values in the range from 4 up to 100.

5. Conclusions and recommendations

The problem of positioning transistors in a "holey cheese" configuration in such a way that the costs are minimized can be approached by first clustering the cells with respect to their interconnectivity, and then positioning the resulting macros into the holes so that the resulting wire length is minimized.

One method to perform the clustering is the so-called Markov Clustering Algorithm. Another method is to use the algorithm developed by Magma to position the cells on a rectangular plane in such a way that the wire length is minimized can be used to carry out a further clustering of the cells. The macros can be obtained by putting a grid with movable grid lines over the rectangle that resulted from the Magma procedure. The exact positioning of the grid lines in such a way that as little wires as possible are cut can be found with the help of a min cut-max flow algorithm, and the resulting grid cells then form the macros.

The positioning of the macros in the holes in such a way that the costs are minimal can now can be translated into a Quadratical Assignment Problem. The problem is then not to find an optimum, i.e. a global minimum of the cost function, but an acceptable local minimum with respect to the computation time. We thought for example of partitioning the macros in the holes randomly, and then make small changes to the partitioning to reduce the wire length until these changes have no more effect. There are numerous approaches to reach the minimal costs with a reasonable computation time. We believe that the method of exchanging successively the positions of two or more macros, starting from a random placement, until no improvement is obtained anymore, is an option worth being looked at. A lot of literature exists both on the hardness of QAPs in general and on the wiring problem as a special case. In particular papers by Anstreicher and Brixius [1],[2],[3] are of interest. They deal with finding true optima, and give references to a host of related material. Most of these are results of the PhD-thesis work of Nathan Brixius [3].

Bibliography

[1] K.M. Anstreicher, N.W., Brixius, (2001), A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming* **89**, 341–357.

[2] K.M. Anstreicher, N.W. Brixius, J.-P. Goux, J. Linderoth (2002, published online 2001), Solving large quadratic assignment problems on computational grids *Mathematical programming* **91** 563–588.

[3] N.W. Brixius, K.M. Anstreicher, The Steinberg Wiring Problem, to appear in *The Sharpest Cut*, M. Grötschel Ed., SIAM.

[4] R.E. Burkard, E. Çela, P.M. Pardalos, L.S. Pitsoulis (1998), *The Quadratic Assignment Problem* Technical Report no. SFB-126, Technische Universität Graz, Mathematik-B.
[5] R. Diestek, *Graph Theory*, 2nd edition, Graduate texts in Mathematics 173, Springer-Verlag, New York, 2000.
[6] S. van Dongen, Graph clustering by flow simulation, PhD. thesis, Universiteit Twente, The Netherlands, 2000.
[7] J. Gross, J. Yellen, *Graph theory and its applications*, CRC Press, Boca Raton, 1999.
[8] D. Jungnickel, *Graphs, Networks and Algorithms, Algorithms and Computation in Mathematics*, Volume 5, Springer, Berlin, 1999.