Exact Combinatorial Optimization Methods for Physical Design of Regular Logic Bricks

Brian Taylor and Larry Pileggi
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{briant, pileggi}@ece.cmu.edu

ABSTRACT

As minimum feature sizes continue to scale down, increasing difficulties with subwavelength lithography have spurred research into more regular layout styles, such as Restrictive Design Rules (RDRs) [11] and regular logic fabrics [10]. In this paper we show that the simplicity and discreteness of regular fabrics give rise to powerful exact combinatorial optimization methods for the brick layout problem (the regular fabric equivalent of the cell layout problem). These methods are either inapplicable or intractable for less regular layout styles, such as the DRC-based approach of standard cell layout. Results from our prototype tool demonstrate that these optimization methods are quite practical for bricks of typical size found in large-scale designs.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Regular Fabrics*; B.7.2 [**Integrated Circuits**]: Placement and Routing.

General Terms

Algorithms, Design.

Keywords

Regularity, Manufacturability, DFM, Combinatorial optimization, Exact methods, Boolean satisfiability.

1. INTRODUCTION

In modern processes, the critical dimension features printed on a chip (e.g., 50 nm for one particular "65 nm" process) are significantly smaller than the illumination wavelength (193 nm for the same process) used to print those features. As a consequence, resolution — the ability to faithfully reproduce the patterns drawn on the mask onto the silicon wafer — is impaired, and Resolution Enhancement Techniques (RETs) such as Optical Proximity Correction (OPC) and Phase Shift Mask (PSM) are employed to bring resolution up to acceptable levels [15]. However, these techniques cannot correct irregular layouts containing arbitrary patterns; for example, OPC often requires space to add serifs, and PSM cannot correct patterns that contain phase conflicts. The intractability of faithful reproduction of irregular, arbitrary layouts has prompted research into more regular, lithography-friendly design styles [12, 10, 11]. While the benefits for manufacturability of layout regularity have been clearly demonstrated [9], the algorithmic implications of the paradigm shift to more regular layout styles have yet to be explored. In this work, we demonstrate that design regularity — which has been necessitated by deep subwavelength lithography challenges — can be exploited for significant improvement at the algorithmic level.

We begin with a discussion of the design methodology assumed in this paper, which was proposed in [10]. In this methodology, the RTL of a design is analyzed and a small set of logic-level netlists (each with a granularity of roughly two to five NAND2 gates) which efficiently 'cover' the design is extracted. Each of these netlists is then mapped to a *regular logic fabric*; the resulting layouts of the logic-level netlists are referred to as *bricks*. By "logic fabric," we refer to the physical structures — diffusion, polysilicon, metal, vias, etc. — with which logic circuits are implemented; hence, *regular logic fabrics* are logic fabrics which are characterized by their regularity. The regular fabric assumed in this paper, which follows that proposed in [10], has the following properties:

- All routing layers (poly and up) are unidirectional. Poly and metal 1 are vertical routing layers, while metal 2 is a horizontal layer.
- Each routing layer has an associated fixed pitch. Moreover, the pitch of each vertical layer is an integral multiple of some *vertical quantum*, and the pitch of each horizontal layer is an integral multiple of some *horizontal quantum*. This, together with the unidirectionality of all layers, implies that brick routing is done on a coarse grid (see Figure 1).
- All PMOS transistors lie in a single row near the top of the brick, and all NMOS transistors lie in a single row near the bottom of the brick (this is akin to the 'single-row' layout style of standard cells).

These restrictions result in the highly regular layout structure illustrated in Figure 1. It has been shown [9] that designs implemented using such a fabric can offer performance

DAC 2007, June 4–8, 2007, San Diego, California, USA.

and area equal to that of standard cell-based designs at the 65nm node. It is expected that regular designs will be able to *exceed* standard cell-based designs in performance and area (and of course yield) at 45nm and below, provided that CAD tools are created to exploit the simplicity and discreteness inherent in regular designs.

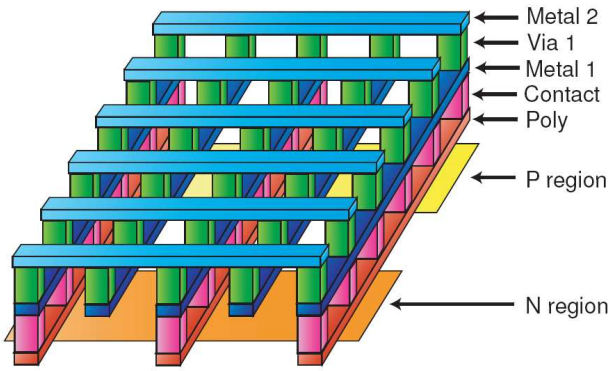

Figure 1: Regular logic fabric.

In this paper, we describe one such CAD tool. Namely, we address the problem of generating brick layouts for the logic-level netlists extracted from the RTL for a given design. The key idea here is that we *exploit the discrete structure of the regular fabric* to obtain exact combinatorial optimization-based layout algorithms which are either inapplicable to or prohibitively expensive for less regular layout styles. Brick layout generation is done in two steps — transistor placement followed by routing — and in each step, the regularity of the logic fabric is leveraged by the layout algorithm. In the transistor placement step, the 'single-row' diffusion style lends itself to highly effective routability metrics, and to an efficient branch and bound algorithm that is optimal with respect to area and strongly Pareto optimal with respect to the routability metrics. Similarly, the discreteness of the coarse routing grid makes feasible a formulation of the brick routing step as a decision problem in the class NP [5]. In turn, this formulation leads to a routing methodology based on Boolean Satisfiability (SAT) that can make strong guarantees of completeness and optimality that virtually no other routing method can make.

The remainder of this paper is organized as follows. In Section 2, we describe our transistor placement algorithm. In Section 3, we formulate the brick routing problem as a decision problem in the class NP, and give a brief overview of an efficient reduction from that decision problem to SAT. In Section 4, we give some promising results based on a prototype layout tool. Finally, in Section 5 we offer some concluding remarks.

2. TRANSISTOR PLACEMENT

Because of the 'single-row' diffusion style of our regular fabric, the transistor placement problem simplifies to the problem of finding a good *ordering* of the PMOS and NMOS transistors within their respective rows. The choice of ordering is very important for two reasons. First, if two transistors that are adjacent in the ordering have a diffusion net in common, then the net may be shared in diffusion between the two transistors, resulting in a brick width reduction of one poly pitch (see Figure 2). Thus, a good ordering will minimize brick width by allowing for as much diffusion sharing between adjacent transistors as possible. The second reason the choice of ordering is important is that some orderings will result in transistor placements that are much easier to route than others. Therefore, a good ordering algorithm should consider the routability of the corresponding placement in addition to the width.

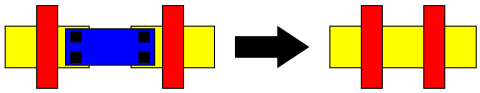

Figure 2: Diffusion sharing between two adjacent transistors.

The problem of finding a good transistor ordering in the 'single-row' diffusion style is referred to as the *transistor chaining* problem, and has been solved in many different ways [14, 1, 7] since it was first proposed in [14]. In this work, we adapt and extend the branch and bound approach of Hwang et al [7]. We chose branch and bound as our optimization strategy because unlike many other transistor chaining algorithms, it allows the exact minimization of width as well as a wide variety of other objectives, such as routability metrics. It is even possible to integrate complex DFM (Design For Manufacturability) considerations, such as STI (Shallow Trench Isolation) stress effects, using such an approach, assuming the availability of good process-dependent models.

In order to ensure that our transistor placements have good routing characteristics as well as minimum width, we have extended Hwang's branch and bound method to exactly minimize two routing metrics: *channel density* and *estimated wirelength*. While the motivation for using wirelength as a routing metric should be obvious, the choice of channel density as a metric warrants further discussion. Given the single-row layout style of our regular fabric, the brick routing problem somewhat resembles the classic channel routing problem (see Figure 3). In the channel routing model, all terminals are clustered at the top and bottom of the routing channel, whereas for the brick routing problem, the terminals are clustered *near* the top and bottom of the channel. The similarity between the brick routing problem and the channel routing problem suggests that channel density is an effective routing metric for bricks; this was in fact confirmed experimentally by comparison with a transistor placement method that minimized only width and estimated wirelength.

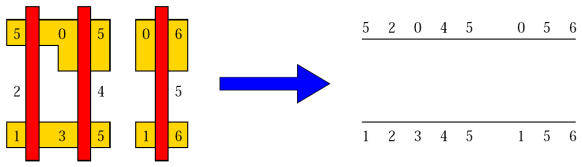

Figure 3: A brick routing instance and its associated channel routing problem.

Our transistor placement methodology is as follows. First, we run Hwang's branch and bound algorithm to find the max number N of abutments possible in any solution. Then, we

run another pass of branch and bound, except that in this pass, we examine all branches of the search tree that lead to orderings with N abutments (rather than just the first such branch). For each of these orderings, we calculate the associated channel density and estimated wirelength, and we choose the ordering of minimal channel density D whose estimated wirelength is minimum among all orderings of channel density D. Thus, we *exactly* minimize brick width, then channel density, then estimated wirelength, in that order. The transistor placements produced by our algorithm are therefore optimal with respect to brick width (and hence area, since brick height is assumed fixed), and strongly Pareto optimal with respect to channel density and estimated wirelength.

3. ROUTING

For a given transistor placement, there are several questions one may ask: Does this placement have a routing? If so, does it have a routing with wirelength at most W? Does it have a routing that uses at most V vias? Does it have a routing that avoids a certain set of undesirable, hard-to-print patterns?

If the answer to any of the above questions is "yes," then there is a short, easily verifiable *proof* that the answer is yes: namely, a routing solution that has the desired properties. Thus, all of the above routing problems — brick routing, minimum wirelengh brick routing, minimum via brick routing, and pattern-prohibited brick routing — are decision problems in the complexity class NP [5], and therefore have polynomial reductions to the Boolean Satisfiability (SAT) problem. In other words, for each instance of any of the aforementioned routing problems, we can generate a boolean formula corresponding to an instance of the SAT problem such that the formula is satisfiable if and only if the instance of the associated routing problem has a solution. Moreover, any solution for the SAT instance can be efficiently "decoded" to obtain a solution for the instance of the associated routing problem.

The choice of SAT as our brick routing engine thus allows us to make strong guarantees that, to our knowledge, virtually no other routing method can make. Unlike heuristic approaches such as maze routing, our brick router *always* finds a routing solution, if one exists. In fact, not only can a SAT-based brick router always find a solution if there is one, it can even find an *optimal* solution, under many different notions of optimality; for example, it is possible to find a routing solution that uses the fewest possible vias. Finally, a SAT-based routing approach facilitates the prohibition of patterns which are deemed undesirable from a lithography point of view.

Although virtually any routing problem can be recast as a boolean satisfiability problem, the general hardness of solving SAT makes satisfiability-based methods effective only for routing problems that can be reduced to *small, highly structured* instances of SAT. This is where *design regularity* comes in: the discreteness of the coarse grid routing structure, which arises from regularity constraints, gives rise to an *efficient* reduction to boolean satisfiability, which we describe in the remainder of this section.

3.1 Reduction to Boolean Satisfiability

The Boolean Satisfiability (SAT) problem is defined in the following way: given a boolen expression F expressed in "product-of-sums" form, also known as Conjunctive Normal Form (CNF), is there an assignment to the variables in the support of F that causes F to evaluate to 1? SAT is NP-Complete [2], which means that any decision problem in the class NP can be efficiently *reduced to* (i.e., expressed as an instance of) Boolean Satisfiability. Despite the suspected hardness of solving SAT in general, SAT solvers have made amazing progress in the last decade, and consequently reduction to SAT is becoming an increasingly popular way to solve design and verification problems in VLSI design.

The general technique for reducing a decision problem in NP to SAT is as follows. First, define a set of boolean variables $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ that efficiently "encodes" the solution space of the original problem. For example, in the brick routing problem, we define a boolean variable x for each wire segment s of the routing grid such that $x = 1$ if and only if s is filled in. Once a suitable set of boolean variables has been defined, specify a set of rules $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ about the properties a solution must have in order to be valid; each such rule R_i is *enforced* by a set of clauses ϕ_i. In the brick routing problem, one such rule might be, "no two distinct nets should ever be shorted together." If we define boolean formula F to be the conjunction of all enforcer clause sets ϕ_i — that is, $F(x_1, x_2, \ldots, x_n) = \phi_1\phi_2\cdots\phi_m$ — then any *satisfying assignment* to F (i.e., any assignment to $x_1, x_2, \ldots, x_n$ which results in $F = 1$) must satisfy *each* rule in $\mathcal{R}$, and hence corresponds to a valid solution to our original problem. The reduction from brick routing to SAT described in this work is just one example of this general technique; similar reductions from physical design problems to SAT can be found in [3]. In fact, the cell layout problem was formulated using SAT in [8]; however, their layout style is much too simplistic for our purposes (e.g., doglegs are not allowed in routes), and would result in much larger layout areas than those produced by our methods.

As the preceding discussion shows, the reduction from the brick routing problem to boolean satisfiability has three components: the set $\mathcal{X}$ of boolean variables that encodes the routing solution space, the set $\mathcal{R}$ of rules which ensure that any satisfying assignment corresponds to a valid routing solution, and the enforcer clause sets ϕ_i corresponding to each rule R_i. Due to space constraints, we cannot fully specify $\mathcal{X}$, $\mathcal{R}$, and the ϕ_i's here; see [13] for the full details of the reduction. Instead, we will list some of the boolean variables, and we will describe an example rule and its corresponding enforcer clauses. We note that a subset of our rules is logically equivalent to (and based upon) the "routability checking" work of [6]. Our reduction to SAT produces $\Theta(rc \lg N)$ clauses, where r is the number of rows of the routing grid, c the number of columns, and N the number of nets. Thus, our reduction is *quasilinear* in N and the size rc of the routing grid.

3.1.1 Variables

We begin by describing two subclasses of the boolean variables in $\mathcal{X}$. The first of these are the *segment variables*, denoted x_{ij}. Variable x_{ij} indicates whether the wire segment of layer x, row i, and column j is filled in. The second kind of variables are the *net ID variables*. Denoted $\vec{x}_{ij}$, these are bit vectors indicating the integer ID of the net passing through the wire segment of layer x, row i, and column j. We refer to the kth bit of $\vec{x}_{ij}$ using the notation x_{ijk}. Note

that the bit vector $\vec{x}_{ij}$ has length b, where b is the number of bits needed to encode a net ID ($b = \lceil \lg N \rceil$, where N is the number of nets).

3.1.2 An Example Rule

The routing rules serve many different purposes. Some serve to ensure that any satisfying assignment corresponds to a completely routed solution; others exist to ensure that the routing obeys design rules; still others prevent patterns that are undesirable from the perspective of DFM, as described in Section 3.3. It is even possible to define rules that limit the number of wire segments used; such rules can be used to *exactly* minimize quantities such as total wirelength or via count, as described in Section 3.2.

As an example of a rule that helps ensure routing correctness, consider the statement, "no two distinct nets should ever be shorted together." This is logically equivalent to the statement, "for every pair of adjacent segments $x_{i_1 j_1}$ and $x_{i_2 j_2}$, if both $x_{i_1 j_1}$ and $x_{i_2 j_2}$ are filled in, then they must not have different net IDs." Symbolically, for every pair of adjacent segments $x_{i_1 j_1}$ and $x_{i_2 j_2}$, we have

$$(x_{i_1 j_1} \wedge x_{i_2 j_2}) \Rightarrow (\vec{x}_{i_1 j_1} = \vec{x}_{i_2 j_2})$$

This can easily be transformed into a propositional formula in "product-of-sums" form, also known as Conjunctive Normal Form (CNF):

$$\bigwedge_{k=0}^{b-1} (\overline{x_{i_1 j_1}} + \overline{x_{i_2 j_2}} + x_{i_1 j_1 k} + \overline{x_{i_2 j_2 k}})(\overline{x_{i_1 j_1}} + \overline{x_{i_2 j_2}} + \overline{x_{i_1 j_1 k}} + x_{i_2 j_2 k})$$

If for every pair of adjacent segments $x_{i_1 j_1}$ and $x_{i_2 j_2}$ we generate the above logic, we get the enforcer clause set ϕ corresponding to our example rule.

3.2 Wirelength Minimization

Suppose that in addition to rules that ensure routing correctness, we enforce a rule of the form, "no more than W wire segments may be filled in." The resulting CNF formula F will then have the property that any satisfying assignment to F will correspond to a routing of total wirelength at most W. Given a lower bound W_ℓ and an upper bound W_u on wirelength, we can *binary search* on W to find the exact minimum $W_{\min}$ for which there is a routing. This is done in the following way: we first try to find a routing with wirelength W at most W_u; if one exists, then we try to find a routing with $W = W_\ell$; if none exists, we then try $W = (W_u + W_\ell)/2$, and then we try either $W = (W_u + W_\ell)/4$ or $W = 3(W_u + W_\ell)/4$ depending on whether there exists a routing with $W = (W_u + W_\ell)/2$, and so on. Clearly this process can be used to find a routing with the minimum possible wirelength using no more than $\lceil \log_2(W_u - W_\ell + 1) \rceil$ SAT solves. Similarly, one can use k processors to search in parallel for $W_{\min}$, and this search will take roughly $T_{\text{avg}} \lceil \log_{k+1}(W_u - W_\ell + 1) \rceil$ seconds, where T_{avg} is the average SAT solve time in seconds.

The rule "no more than W wire segments may be filled in" is logically equivalent to the expression $\sum_i w_i \leq W$, where summation is done over all segment variables w_i that correspond to wire segments. In order to convert this expression into CNF logic, we describe a circuit that first sums the wire segments w_i and then compares the sum to W. Once a circuit with this functionality is known, it is trivial to emulate the circuit using CNF logic. An arbitrary combinational circuit that computes $f(x_1, \ldots, x_n)$ and whose output is Z can be emulated by converting the expression $Z \overline{\oplus} f(x_1, \ldots, x_n)$ to CNF form; the resulting CNF formula can only be satisfied by an assignment that sets Z equal to $f(x_1, \ldots, x_n)$.

It is quite easy to generate CNF logic to check whether a bit vector is at most some constant W (see pp. 17–18 of [13]), so here we will only describe how to efficiently construct a circuit (shown in Figure 4) that adds the n boolean variables corresponding to wire segments.

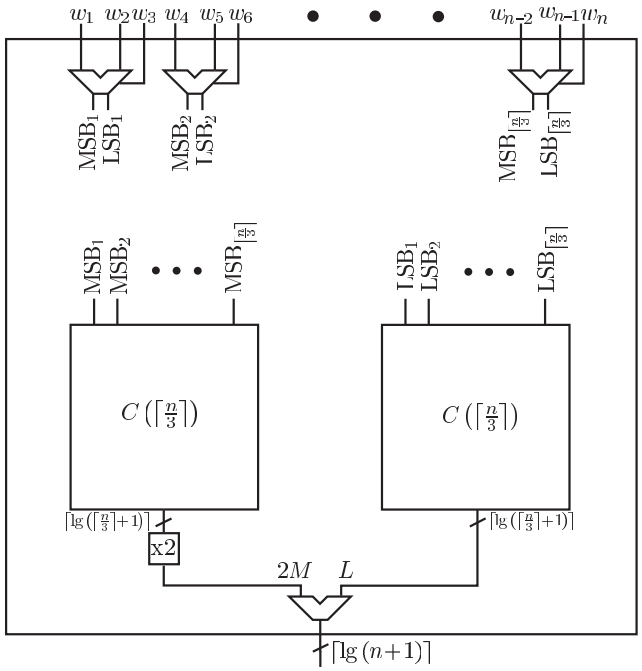

Figure 4: The n-bit divide-and-conquer addition circuit $C(n)$.

The addition circuit consists of three stages. In the first stage, we form $\lceil n/3 \rceil$ 2-bit sums; let MSB_i and LSB_i denote the MSB and the LSB of the ith sum. In the second stage, we recursively add all the MSBs and all the LSBs; let these sums be denoted by M and L, respectively. The final output sum is equal to

$$\sum_{i=1}^{\lceil n/3 \rceil} (2 \cdot \text{MSB}_i + \text{LSB}_i) = 2 \sum_{i=1}^{\lceil n/3 \rceil} \text{MSB}_i + \sum_{i=1}^{\lceil n/3 \rceil} \text{LSB}_i = 2M + L,$$

so in the final stage, we add $2M$ to L.

The reason we use this circuit topology is that it is *area-efficient*; the less hardware we have to emulate, the smaller our resulting CNF formula will be. Both the number of variables and the number of clauses needed to implement this circuit are directly proportional to the number of 1-bit adders in the circuit. Looking at Figure 4, we can write the following recurrence relation for the number of adders $A(n)$ used by a divide-and-conquer circuit that adds n 1-bit numbers:

$$A(n) = \lceil n/3 \rceil + 2A(\lceil n/3 \rceil) + \lceil \lg(\lceil n/3 \rceil + 1) \rceil$$

Solving this recurrence, we get $A(n) = \Theta(n - n^{\log_3 2}) \approx \Theta(n - n^{0.631})$; thus, the number of 1-bit adders needed (and hence, the number of clauses needed) to compute the sum is linear in the number of bits added. Moreover, the subtractive sublinear $n^{\log_3 2}$ term makes this topology asymp-

totically superior to more naïve adder circuits, such as the binary tree adder.

3.3 DFM-Aware Pattern Prohibition

From a DFM (Design for Manufacturability) perspective, one of the most attractive features of our SAT-based routing methodology is the ability to explicitly *prohibit* patterns which are difficult to print using subwavelength lithography. As discussed in Section 1, Resolution Enhancement Techniques (RETs) cannot correct any arbitrary pattern; thus, it is necessary to avoid those patterns that are difficult or impossible to correct. An example of a pattern in the metal 2 (M2) layer that cannot be corrected by OPC is shown in Figure 5. When two horizontally adjacent M2 segments at minimum separation distance are flanked above and below by unbroken M2 strips, it becomes impossible to apply the "hammerhead" OPC serifs to the M2 segments required to prevent line-end pullback.

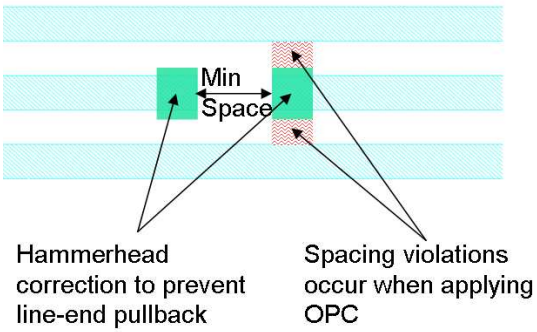

Figure 5: An M2 pattern that cannot be corrected by OPC.

Importantly, our SAT formulation of the routing problem can easily and efficiently prohibit such patterns explicitly. In our abstraction of the routing grid, the pattern of Figure 5 corresponds to the grid segment pattern shown in Figure 6. If $M2_{ij}$ is the segment variable indicating whether the segment in row i and column j of the M2 layer is filled in, then we want to avoid the case where $M2_{30} = M2_{31} = M2_{32} = M2_{33} = M2_{20} = M2_{23} = M2_{10} = M2_{11} = M2_{12} = M2_{13} = 1$ and $M2_{21} = M2_{22} = 0$. In other words, we want the following clause to be satisfied:

$$\overline{M2_{30}} + \overline{M2_{31}} + \overline{M2_{32}} + \overline{M2_{33}} + \overline{M2_{20}} + \overline{M2_{23}} + \overline{M2_{10}} + \overline{M2_{11}} + \overline{M2_{12}} + \overline{M2_{13}} + M2_{21} + M2_{22}$$

Thus, in order to prohibit a particular pattern, we need just one clause for each position where that pattern could occur. Clearly, the number of possible positions for a given pattern is bounded by the size rc of the grid (where r is the number of rows and c the number of columns), so the number of clauses needed to prohibit any particular pattern is *linear* in the size of the grid.

4. EXPERIMENTAL RESULTS

We have developed a prototype brick layout tool that implements the transistor placement and routing algorithms described in this paper. Table 1 contains results from using this tool to lay out five benchmark bricks of various size. From left to right, the columns contain benchmark names,

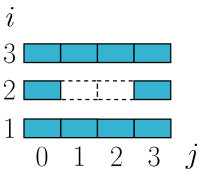

Figure 6: The grid segment pattern corresponding to the unwanted M2 pattern of Figure 5.

transistor counts, number of variables/clauses in the SAT routing formula, runtime in seconds of the entire flow when no wirelength minimization is done, and runtime in seconds when exact via minimization is done. We emphasize that the runtimes are for *the entire layout flow*, not just for routing. Experiments were run on an Intel P4 2.8 GHz machine running GNU/Linux. We used `minisat2` as our SAT solver [4]. We would like to have compared our prototype tool against commercial cell layout tools, but unfortunately we are unaware of any available tools that allow the specification of the regularity constraints (fixed pitch, one direction per layer, and so on) that define our regular fabric, or that allow for the explicit prohibition of certain patterns.

name	xtors	# vars	# clauses	no min	via min
ex1	8	3259	19613	.62	.64
ex2	10	4791	26873	.87	1.7
ex3	12	6905	44903	1.4	2.9
ex4	16	8809	57770	2.3	6.0
ex5	18	11667	83075	6.5	9.2

Table 1: Results from five benchmark bricks.

As discussed in Section 1, in the design methodology assumed in this paper, a design is mapped to a small set of logic-level netlists; each of these netlists (whose layouts we refer to as 'bricks') has a granularity of roughly two to five NAND2 gates (that is, its transistor count is limited to roughly 20). The reason for this size limitation on bricks is as follows. Since the entire design is implemented using a *small* number of bricks (typically between 20 and 30), it is important that the individual bricks not be too large, or else the design will suffer from poor area utilization. Thus, the benchmark bricks of Table 1 range in size from small to large, with the largest being near the upper bound imposed by utilization considerations.

As is clear from the runtimes, the brick layout methods proposed in this paper are quite practical — no benchmark took more than a minute. It should be noted that SAT solving runtimes have notoriously large variance, so the above numbers are best interpreted as an order-of-magnitude estimate for benchmarks of similar size. The time reported for via minimization is not the time taken by the entire binary search, but rather the time for the final SAT solve with the via bound equal to the minimum possible. However, it can be shown that with a few processors searching in parallel for the minimum number of vias possible, the expected total runtime is a small factor larger than the average SAT solve time T_{avg} (for example, with $k = 3$ processors the expected runtime is less than $4T_{\text{avg}}$). Thus, the expected total via minimization time should only be a few times larger than the reported number, which itself should be regarded as an order-of-magnitude estimate.

We have observed that via minimization is far more time-efficient than wirelength minimization; exact via minimization generally takes seconds, whereas exact wirelength minimization can take minutes or even hours. However, exact via minimization usually produces layouts of minimum or near-minimum wirelength, as shown in Figure 7. Thus, it suffices for practical purposes to do via minimization rather than the far more time-consuming wirelength minimization. This is in fact yet *another* benefit of design regularity! More precisely, this is a benefit of unidirectionality: since all routing layers are unidirectional, and since metal 2 is the only horizontal routing layer, a given route must use unnecessary vias to meander. Minimizing the number of allowed vias therefore prevents routes from meandering, and thus indirectly minimizes total wirelength.

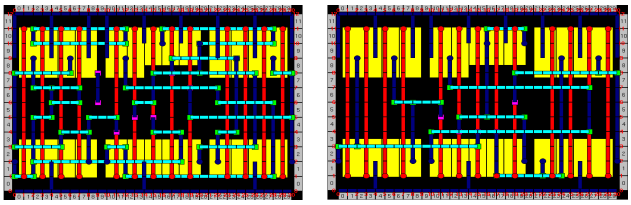

Figure 7: Left: no WL minimization. Right: via-minimized layout.

5. CONCLUSIONS AND FUTURE WORK

As our experimental results show, design regularity — which has heretofore been championed for purely manufacturability-related reasons — is actually desirable from an algorithmic point of view as well. The problems of brick-level transistor placement and routing, under the constraints of design regularity, take on a discrete character that is amenable to exact methods of combinatorial optimization such as branch and bound and SAT. Moreover, as we descend deeper into the subwavelength regime, design regularity becomes more important, while the heuristic methods driving combinational optimization engines such as SAT grow ever more powerful. Thus, the approach of solving physical design problems under strict regularity constraints via exact algorithms such as SAT will only grow more attractive in time. It will be interesting to see if other, perhaps more large-scale physical design problems under regularity constraints can be solved using similar techniques in the future.

This work was supported in part by the Semiconductor Research Corporation via a student fellowship. We would like to thank Padmini Gopalakrishnan and Slava Rovner for many helpful discussions related to this work.

6. REFERENCES

[1] R. Bar-Yehuda, J. A. Feldman, R. Y. Pinter, and S. Wimer. Depth-First Search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation. *IEEE Transactions on Computer-Aided Design*, 8(7):737–743, 1989.

[2] S. A. Cook. The Complexity of Theorem Proving Procedures. *Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[3] S. Devadas. Optimal Layout Via Boolean Satisfiability. *ICCAD-89*, pages 294–297, 1989.

[4] N. Eén and N. Sörensen. An Extensible SAT-solver. *In Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 502–518, 2003.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[6] W. N. N. Hung, X. Song, T. Kam, L. Cheng, and G. Yang. Routability Checking for Three-Dimensional Architectures. *IEEE Transactions on VLSI Systems*, pages 1371–1374, 2004.

[7] C.-Y. Hwang, Y.-C. Hsieh, Y.-L. Lin, and Y.-C. Hsu. An Optimal Transistor Chaining Algorithm for CMOS Cell Layout. *ICCAD-89*, pages 344–347, 1989.

[8] T. Iizuka, M. Ikeda, and K. Asada. High-Speed Layout Synthesis for Minimum-Width CMOS Logic Cells via Boolean Satisfiability. *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 149–154, 2004.

[9] T. Jhaveri, L. Pileggi, V. Rovner, and A. Strojwas. Maximization of Layout Printability/Manufacturability by Extreme Layout Regularity. *In Proceedings of SPIE*, Vol. 6156, 2006.

[10] V. Kheterpal, V. V. Rovner, T. G. Hersan, D. Motiani, Y. Takegawa, A. J. Strojwas, and L. Pileggi. Design Methodology for IC Manufacturability Based on Regular Logic Bricks. *In Proceedings of the 42nd Conference on Design Automation*, pages 353–358, 2005.

[11] M. Lavin, F.-L. Heng, and G. Northrop. Backend CAD Flows for "Restrictive Design Rules". *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, pages 739–746, 2004.

[12] Y. Ran and M. Marek-Sadowska. The Magic of a Via-Configurable Regular Fabric. *International Conference on Computer Design*, pages 338–343, 2004.

[13] B. Taylor. Automated Layout of Regular Fabric Bricks. Master's thesis, Carnegie Mellon University, 2005.

[14] T. Uehara and W. VanCleemput. Optimal Layout of CMOS Functional Arrays. *IEEE Transactions on Computers*, C30:305–314, 1981.

[15] B. Wong, A. Mittal, Y. Cao, and C. Starr. *Nano-CMOS Circuit and Physical Design*. John Wiley and Sons, 2005.