Synthetic Benchmark Circuits for Timing-driven Physical Design Applications

P. Verplaetse D. Stroobandt* J. Van Campenhout
Department of Electronics and Information Systems
Ghent University
St-Pietersnieuwstraat 41
B-9000 Ghent, Belgium

Abstract *For the development and evaluation of new algorithms, architectures and technologies, a huge amount of benchmark circuits with suitable characteristic parameters are required. Synthetic circuits are a viable alternative to real circuits for compiling benchmark suites. A major advantage of synthetic benchmark circuits is that full control of the important parameters is provided. In this paper, an existing netlist generation algorithm based on bottom-up clustering of subcircuits according to Rent's rule is extended to generate circuits that are more realistic than before. The stochastic properties of the Rent behavior are taken into account, and improvements have been made to increase the accuracy of the imposed Rent characteristics. This guarantees a realistic structure of the interconnection topology, which can be adjusted in a controlled manner. A scheme for combinational loop prevention has been augmented with a delay control mechanism, such that they are truly suitable for timing-driven applications. An indirect validation approach is used to verify that existing placement algorithms exhibit comparable behavior for both real and synthetic circuits.*

Keywords: Synthetic benchmarks, physical design, Rent's rule, timing-driven.

1 Introduction

With the continuing evolution of VLSI technology, more and more gates are being pushed on a single chip. The design of chips becomes more complicated, and the problem is worsened by deep-submicron effects due to the ever shrinking feature size. Computer-aided design tools have become indispensable to cope with the complexity and the limited time resources. The development and evaluation of new technologies, architectures and electronic design automation (EDA) tools requires a large number of benchmark circuits. A benchmark suite is a set of circuits that – in the ideal case – is representative for circuits at which the EDA tool, technology or architecture is aimed. Benchmark suites form a standard comparison basis that is accepted throughout the EDA community.

Initiatives for distributing benchmark circuits have been taken [1, 2, 3]. However, the existing benchmark suites are often not sufficient, since they usually consist of too few and too small circuits. Because of the proprietary nature of industrial circuits, it is almost impossible to compile sufficiently large sets of sufficiently large realistic circuits. Recently, the generation of synthetic benchmark circuits is becoming to be recognized as a viable alternative.

The major advantage of synthetic circuits is that they provide full control of the important characteristic parameters, such as circuit size, interconnection structure and functionality. Ideally, those parameters can be set independently, and one has full control over the granularity of the synthetic benchmark suites. For example, one can generate a set of circuits with increasing circuit size while maintaining the pin count. This allows a careful analysis of the impact of specific circuit parameters on the behavior of an algorithm or architecture, which would be nearly impossible to realize with a set of real benchmark circuits. It is also possible to generate circuits for which certain properties hold by construction, e.g., Krishnamurthy [4] describes a method for generating circuits with a known minimal solution to the NP-complete partitioning problem.

An overview of various existing methods for synthetic benchmark generation is presented in [5]. It seems impossible to efficiently generate synthetic circuits that capture all aspects of real circuits. All approaches focus on either the interconnection structure or the functionality of the circuit. Of course, it would be possible to describe a set of algorithms in a high-level description language, and use logic optimization tools to generate a set of logic-level circuits, but this would defy the purpose of synthetic benchmarks, since this method is very time-consuming, and no full control over the circuit parameters can be provided.

In this paper we focus on synthetic benchmark circuits for physical design applications. For these applications, the functionality of the design is of secondary importance. Such applications include (but are not limited to) the development and evaluation of algorithms for floorplanning, placement and routing, and the development and evaluation of new architectures and technologies. Since

*Dirk Stroobandt is a Post-doctoral Fellow of the Fund for Scientific Research – Flanders (Belgium)(F.W.O.).

timing and design closure is becoming increasingly important, timing-driven methods and techniques such as automatic repeater insertion, buffer optimization or even local logic optimization are being integrated with the standard physical design algorithms. To allow benchmarks to be suitable for these applications, both the interconnection topology and the delay properties must be modeled properly.

Of all methods presented in [5], the approach based on bottom-up clustering of subcircuits according to Rent's rule [6] appears to be the most promising. Since Rent's rule is followed during generation, a realistic structure of the interconnection topology is guaranteed and can be adjusted in a controlled manner. However, the method still has a few shortcomings: though a combinational loop prevention scheme has been implemented, the maximum path length (which corresponds to the delay in a unit-delay model) is ignored completely. This results in excessive delays, which makes it unusable for timing-driven applications. Also, the circuits are very regular, and the stochastic properties of the interconnection topology [7, 8] were not modeled an a realistic manner. In this paper, a new version of the netlist generation algorithm is presented to overcome these issues. The algorithm for the generation of netlists is presented in section 2. The adjustments required to make the netlists suitable for timing-driven applications are described in section 3. Section 4 addresses the important topic of validation of the synthetic benchmark circuits. Finally, some concluding remarks are presented in section 5.

2 Net generation according to Rent's rule

For physical design applications, the principal characteristic parameter to be taken into account is the interconnection structure. The critical part in our benchmark generation process is the net generation. The net generation process is based on (bottom-up) recursive clustering, which is the opposite of (top-down) recursive partitioning. During net generation a certain Rent behavior is aimed at. This allows control of the complexity of the interconnection topology.

2.1 Rent's rule

When partitioning a circuit into more or less equally sized modules (subcircuits), constrained to a certain terminal minimization objective, there seems to be a power-law relationship between the average terminal count T and the module size B. This empirical relationship is known as *Rent's rule* [9]:

$$T = kB^p \tag{1}$$

The parameter k is known as the *Rent coefficient*, and p is the *Rent exponent*. The latter is correlated to the complexity of the interconnection topology, since circuits with a higher Rent exponent have more global interconnections, and thus tend to have more long wires after placement and routing. For big module sizes, there might be some deviation to Rent's rule, which is known as *region II* of Rent's rule. This can be modeled with a different Rent exponent. When a circuit can be generated that follows Rent's rule by construction, it will be guaranteed to have a realistic interconnection structure.

Rent's rule states something about the sample mean of the terminal count distribution for a set of modules in a certain partition of the circuit. This is only a first-order statistic. It has been shown in [6] that the second-order statistics are an important property of the circuit topology, as they define the irregularity of the circuit. A theoretical expression for the sample variance of the terminal count distribution has been derived in [7, 8]. This quite complex expression is a sum of exponential terms similar to (1). For the purpose of synthetic benchmark generation, it is sufficient to use a more simplified version by retaining only a single term:

$$\sigma_T = \sigma_0 B^q \tag{2}$$

Theoretically, q should equal p, but in practice the exponent for the standard deviation of the terminal count distribution is usually somewhat smaller than the Rent exponent.

The terminal count distribution can be approximated as a normal distribution, hence it is sufficient to model the sample mean and standard deviation. However, for benchmark generation the balance between the number of inputs I and outputs O is also required. Based on the assumption of a power-law net-degree distribution, a theoretical expression for the output fraction $g = O/T = O/(I+O)$ can be derived [10]. However, due to irregularities and other deviations in real circuits, it seems to be more efficient to use the following empirical relationship:

$$g = a + b\log(B) \tag{3}$$

When inspecting the output fraction for the different modules of a single partition, more variation is observed than can be expected from the small variations of the module size B. As a simple but sufficient approximation, the output fraction can be modeled as a normal distribution with mean given by (3) and a fixed standard deviation. Typical values for the mean output fraction are between 0.3 and 0.5; the standard deviation can be as high as 0.15.

2.2 Recursive clustering

One way of determining the Rent exponent is by bipartitioning the circuit and then recursively bipartitioning the resulting modules, until the complete circuit is partitioned into single gates. The partitions that are obtained during this recursive bipartitioning process will satisfy the terminal minimization constraint when a mincut or ratiocut algorithm is used during bipartitioning. Because of the recursive partitioning scheme, a hierarchy between the different modules exists. Each module has a parent module (except the complete circuit) and two child modules (except the gates).

This top-down recursive partitioning scheme has been used as a basis for a synthetic benchmark generation method by Darnauer and Dai [11]. The number of inputs and outputs at the top level is set to a user specified value. At this point the circuit has a specified number of gates, but no nets have been defined. Then the circuit is partitioned, and the nets that would have been cut at that level are being assigned to the circuit. This process is repeated until the remaining parts consist of single gates. Because of the top-down approach, it is difficult to have good control on the fanout of the individual gates. It is also not trivial to guarantee that no combinational loops are created.

The reverse process of recursive partitioning is the *recursive clustering* of modules. By using this bottom-up approach for synthetic benchmark generation, none of the above problems exist. One starts with a set of gates, which are the building blocks of the circuit. The number of inputs and outputs can be chosen freely for each gate. Then the gates are combined into modules, and the modules are further combined until only a single module remains, which is the resulting circuit.

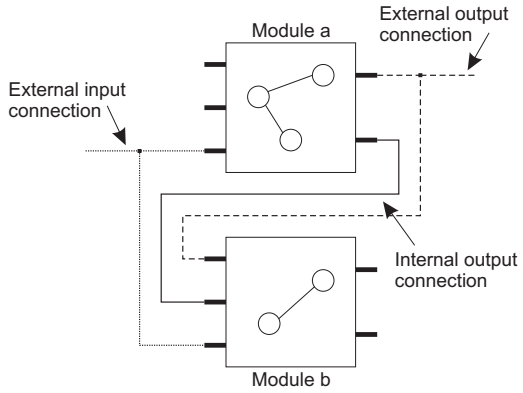

Figure 1: The different types of connections during net generation.

The combination of gates into modules is performed by connecting nets. This process is called *net generation*. Consider the combination of two modules `a` and `b` in a resulting module `c` (fig. 1). Different types of connections are possible. An *internal net* connects two terminals of both modules. The net is not available at higher levels of the hierarchy. An *external net* also connects two terminals, but provides the net to the next level by adding a terminal to module `c`. An *output connection* is a connection from an output terminal of one module to an input terminal of the other module. Such a connection can be internal or external; in the latter case an output is added to the resulting module. An *input connection* is a connection between inputs of both modules. This connection must always be external since the input must be made available at the next level to provide the net with a driver.

Module `c` contains $B_c = B_a + B_b$ gates. The total number of inputs and outputs before combination is $I_a + I_b$ and $O_a + O_b$ respectively. An external connection eliminates exactly one input, while an internal connection eliminates both an input and an output. Denote the number of internal and external connections as S_i and S_e respectively, then the resulting number of inputs and outputs is given by

$$I_c = I_a + I_b - S_i - S_e \quad (4)$$

$$O_c = O_a + O_b - S_i \quad (5)$$

The algorithm for netlist generation by recursive clustering consists of the following steps:

1. Select a set of gates according to the specified library and cell distribution. Put each gate into a module.
2. Select two modules. Calculate the mean and standard deviation of T and g with expressions (1,2,3) and sample a random T and g from the normal distributions with these parameters. This can be converted to the target number of inputs I_c and outputs O_c. Calculate the required number of internal and external connections with expressions (4,5).
3. Iterate over all outputs of module `a` and inputs of module `b` and simultaneously over all outputs of module `b` and inputs of module `a`. Choose the direction (`a` to `b` or `b` to `a`) at random and make an external output connection if allowed (see section 3). Make sure no more than $S_i + S_e$ connections are made.
4. Convert S_i of the previously made connections to internal connections. Of course, this is only possible if at least S_i connections were made in the previous step.
5. Iterate over all inputs of both modules and make external input connections until the total number of external connections equals S_e.
6. Repeat from step 2 until only one single module remains.

The result of this algorithm is a homogeneous synthetic benchmark circuit. *Hierarchical heterogeneity*, i.e. multiple Rent regions, can easily be implemented by using different parameters in expressions (1,2,3) for each region. *Spatial heterogeneity*, i.e. a circuit that consists of multiple parts with different complexity, requires the generation of multiple sub-circuits with different Rent parameters, that are further combined into a single circuit. This can be achieved by multi-level generation: during netlist generation, the building blocks do not need to be library cells, but can also be macrocells that are by themselves synthetic circuits with specific Rent parameters. Hierarchically structures of unlimited complexity can be generated this way, though in general two levels of hierarchy are sufficient to generate realistic spatially heterogeneous circuits.

Applying the algorithm as described above can be very memory consuming, especially as the data structures for

the modules become bigger (see section 3), since the hierarchy tree is built in a *breadth-first* manner. This can be avoided by first building the hierarchy tree without actually creating or combining the modules, and then creating all the modules recursively with a *depth-first* approach. This is possible since the selection of the modules is at random or depends on the module size (which can be calculated beforehand) but is independent from the number of terminals (which is only known after combining the modules).

3 Timing-driven applications

When allowing all connections in step 3 of the previous algorithm, the circuits that are generated may have combinational loops. This may not be important for classic physical design algorithms, but it is definitely an issue for timing-driven applications. Many algorithms will simply choke on circuits that have combinational loops. First we will address the problem of combinational loop prevention. Then we will discuss delay control.

3.1 Combinational loop prevention

In order to prevent combinational loops from being created, extra data must be stored while generating the netlist to decide whether a connection can be made without introducing combinational loops. One possibility is to remember for each input the set of outputs that can be controlled through a combinational path (OCC-set). Input connections can always be made, but output connections must be checked first. Consider for example the possible output connections from module `a` to module `b` in fig. 2. Since there already was an output connection from 8 to 2, connection 4 to 6 is not allowed. This is reflected in the OCC-set of 6={7,8,4}.

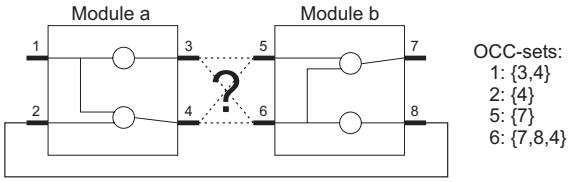

Figure 2: Combinational loop prevention based on OCC-sets.

Step 1 in the net generation algorithm needs to be adjusted to initialize the OCC-sets: for combinational gates the OCC-sets are initialized with all outputs; for sequential gates the OCC-sets are empty. After making a connection, the OCC-sets must be updated. For an input connection (step 5) between inputs `IA` and `IB`, the OCC-sets of the two inputs, must be merged. For an output connection (step 3) from `OA` to `IB`, the OCC-set of input `IB` must be added to the OCC-sets of all the inputs that have output `OA` in their OCC-set.

Of course, other data structures may be used. The disadvantage of this data structure is that one must search through the OCC-sets of all inputs after an output connections is made. One may consider using a redundant data structure, where next to the OCC-sets, for each output the set of inputs that is observable through a combinational path (IOC-set) is maintained. However, it is our experience that this does not improve the performance. On the contrary: the search operation becomes faster, but the possible speed-up is lost due to the overhead for maintaining a more complex data structure.

3.2 Maximum path length

Though the previous approach guarantees that the circuits are free from combinational loops, they are still very unrealistic because extremely long paths may occur. We measured a maximum delay of 1736 gates on a medium sized synthetic circuit (9000 gates, of which 1000 were sequential). Considering the fact that typical path lengths are in the order of magnitude of 30 gates, this is unacceptable. In addition to combinational loop checking, it is also necessary to check if the path length of the created nets is not too long. Again different solutions are possible. Our solution is based on the OCC-sets defined above. For each output in an OCC-set the length of the maximum combinational path from the input to the output is stored as well.

In addition, for each output the maximum delay from any input or any internal sequential gate (MOD) is stored. During net generation, the maximum allowed delay that can be added to each input (MAID) is maintained. A connection is only allowed when the delay constraint for the input is not violated. Again, input connections can be made without restriction, since this does not increase any path lengths.

The different steps in the net generation process have to be adjusted. During step 1, the OCC-sets, MAIDs and MODs are initialized. Each gate has a specific combinational delay D_c (for a unit delay model, $D_c = 1$). The MODs of the outputs are set to this delay. If the gate is combinational, the OCC-delays are set to D_c as well. To ensure that the maximum path length of the circuit to be generated will not exceed a certain value M, it is sufficient to set all the MAIDs of the inputs of combinational gates to $M - D_c$, and those of the sequential gates to M.

For an input connection (step 5), the MAID of the new input is set to the minimum of the MAIDs of the inputs that have been combined. For an output connection (step 3), the situation is a little more complicated. First, the MODs of the outputs in the OCC-set of input `IB` are adjusted if they are exceeded by the new path length to that output, which is MOD(`OA`)+OCC-delay. Second, for all inputs that have output `OA` in their OCC-set, the OCC-set of input `IB` is added to their OCC-set, but with OCC-delay equal to the sum of the OCC-delay from that input to `OA` and the OCC-delay from `IB` to the output in the OCC-set of `IB` (unless that output already existed in the OCC-set of that input and the OCC-delay is already longer). Third, the MAIDs of the inputs that have output `OA` in their OCC-set are adjusted if the new constraint

(MAID(IB)-OCC-delay) is more restricting.

Though in theory the above extension of the algorithm ensures that no paths longer than M are being generated, the algorithm does not perform well in practice. The maximum path length M is reached after only a few combinations, and the algorithm has a hard time finding allowable connections at the highest levels of the hierarchy. As a result, many inputs and outputs cannot be eliminated, and the resulting number of primary inputs and outputs usually exceeds the target value.

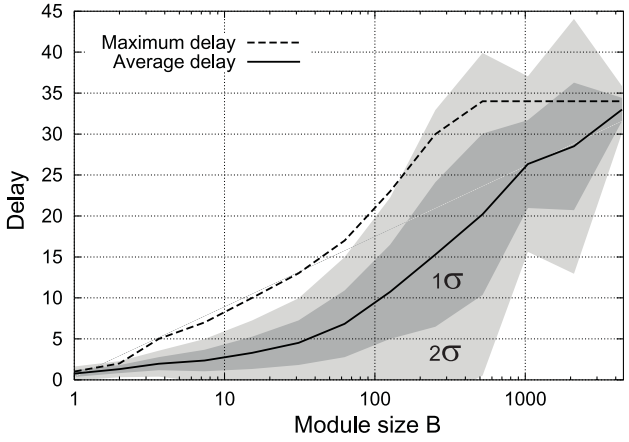

Figure 3: The maximum delay during recursive partitioning of a circuit.

Fig. 3 shows the average and maximum delay for partitions of a real circuit into modules with approximately B gates. For sufficiently small module sizes (less than 70% of the circuit size on a logarithmic scale), the maximum delay appears to decrease almost linear with $\log(B)$. When scaling the MAIDs with a similar size factor γ,

$$\gamma = \begin{cases} \frac{\log(B)}{\log(B_{co})} & : \quad B < B_{co} \\ 1 & : \quad B \geq B_{co} \end{cases} \tag{6}$$

the net generation process produces good results for a broad range of cut-off module sizes B_{co}.

3.3 Delay distribution

The delay distribution (maximum path delay for all inputs of sequential gates and all primary outputs) for a circuit with 9000 gates (of which 1000 are combinational cells), $B_{co} = 70\%$ and maximum delay $M = 40$ is shown in fig. 4(a). The delay distribution corresponds to that of a circuit that is extremely optimized. Typical for these circuits is the high amount of paths with near maximum delay.

The actual delay distribution can be controlled by adjusting the maximum delay value M which is used in step 1 to calculate the MAIDs for each gate. By sampling these values from a distribution $\mathcal{D}_M$, different delay distributions can be obtained depending on the shape of $\mathcal{D}_M$. This allows the generation of circuits that appear to be less optimized, such as fig. 4(b).

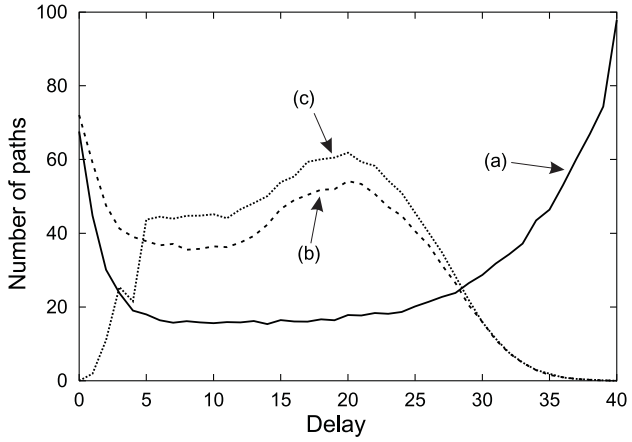

Figure 4: Different delay distributions that can be obtained.

Note that all circuits have near identical distributions for small delay values. The number of paths is determined by the probability that such a path is formed by the selection process in step 2 and the choice of inputs and outputs that is combined in steps 3 and 5. For small values of path length, these probabilities are mainly determined by the choices made at the lowest levels of the hierarchy. A pure random method is applied, hence the number of paths for small delay values is mainly determined by the fraction of sequential gates. To reduce the number of paths for small delay values, it is sufficient to introduce a minimum delay for all sequential gates, and to prevent clustering of these gates with modules that are too small. This is shown in fig. 4(c).

4 Validation

An important but often forgotten issue with synthetic benchmark generation is the validation of the synthetic benchmark suites: one needs to verify that the synthetic circuits exhibit properties that are similar to those of real circuits. Two different approaches can be identified [5]:

Direct validation This implies measuring the circuit characteristics directly and comparing them to those of real circuits. Direct validation is a very strict validation method, and requires that all aspects of the circuit are analyzed. Usually, not all aspects of real circuits are taken into account for the generation of synthetic benchmarks. Therefore, such circuits will not pass as realistic. This is also the case for circuits generated by the algorithm described in this paper: the circuits have a realistic interconnection topology, but the functionality of the circuits is completely ignored. The synthetic circuits appear to be very redundant [6], which makes them inappropriate for most logic design applications. Nevertheless, direct validation can still provide valuable insight in the potential and limitations of synthetic benchmark approaches.

Indirect validation This is a more relaxed validation method, where the efficiency of a set of algorithms of the same type is evaluated with both real and synthetic suites. For a good benchmark suite, the results will be comparable. Though it is impossible to prove that the synthetic benchmarks generated by recursive clustering circuits are realistic in all aspects – the circuits are too redundant – it can be shown by indirect validation that these circuits can be applied for benchmarking of physical design applications.

4.1 Direct validation

As a first step towards the direct validation of the synthetic benchmarks, we cloned a subset of the ISPD98 benchmark suite [2]. The Rent characteristics of circuits `ibm01` to `ibm10` were measured by recursively bipartitioning the circuits with hMetis [12]. For each circuit, a synthetic counterpart was generated with the same target Rent characteristic. The gate distribution for the synthetic circuits was derived from the original circuits. Since the ISPD98 benchmark circuits do not provide functional or delay information, we arbitrarily set the number of sequential gates to 10% of the original gate count. The circuits were generated with a maximum path length ranging from 100 gates for `ibm01` to 150 gates for `ibm10`.

Table 1: Number of inserted flip-flops and average wire lengths for the cloned ISPD98 circuits.

Circuit	ISPD98		Clones	
	gates	$\overline{l}$	flops (%)	$\overline{l}$ (%)
ibm01	12506	10.07	3.65	101.30
ibm02	19342	18.30	3.68	89.722
ibm03	22853	15.93	5.06	95.029
ibm04	27220	15.63	3.09	103.32
ibm05	28146	31.77	3.86	95.51
ibm06	32332	16.74	4.13	101.00
ibm07	45639	17.16	3.23	110.71
ibm08	51022	19.07	3.90	106.83
ibm09	53110	14.71	3.17	111.25
ibm10	68685	21.48	4.19	111.54

Due to the various correction techniques described in the previous section, the synthetic circuits exhibit Rent characteristics that are nearly identical to their original counterparts. This level of accuracy could only be achieved by inserting flip-flops during the net generation process. The fourth column of table 1 shows how many flip flops (as a percentage of the original gate count) had to be inserted. It appears that in general about 4% of the original number of gates has to be inserted.

Since the synthetic benchmark generation method is targeted to physical design applications, we placed the circuits with a commercially available placement tool (QPlace v5.0.23), and measured the average wire length after placement. The average wire lengths for the original circuits are displayed in the third column of table 1; the average wire lengths of the synthetic counterparts are shown in the last column (relative to those of the original circuits). The average wire length of the clones is always within 12% of the original average wire-length. Fig. 5 shows the wire length distributions for `ibm10`, the circuit with the worst. Even for this circuit the wire length distributions appear relatively close together. This indicates that the interconnection topology is well characterized by the Rent properties of the circuits, and that the synthetic circuits seem realistic for physical design applications such as placement. This is, however, only an indication, and an indirect validation approach is required to draw concrete conclusions regarding the suitability of the synthetic circuits.

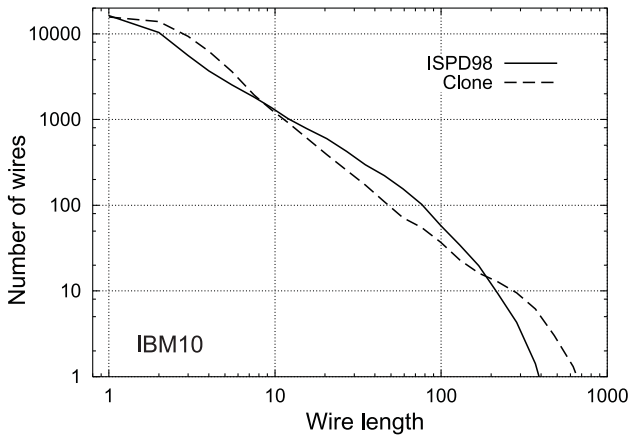

Figure 5: Wire length distributions of circuit `ibm10` and the synthetic counterpart.

4.2 Indirect validation

For synthetic circuits to be a viable alternative for real circuits for physical design applications, it is sufficient to show that comparable results are obtained from physical design experiments with both real and synthetic benchmark suites. For the indirect validation of the synthetic benchmarks, we compared the QPlace placement tool with two other placement algorithms: Plato and SA. The former is a hierarchical placement approach based on quadrisection; the latter is a flat placement algorithm based on simulated annealing with a very slow cooling schedule. Fig. 6 shows the average wire-lengths for all three placement approaches, measured with both the real benchmark suite and the synthetic counterparts.

Note that this figure is not intended to compare the different algorithms against each other. This would be a rather unfair comparison, since the QPlace algorithm exhibits much smaller run times than Plato (which is about 5 times slower), and the SA approach is extremely slow compared to the other algorithms (run times can be over a thousand times slower than QPlace for circuits of this

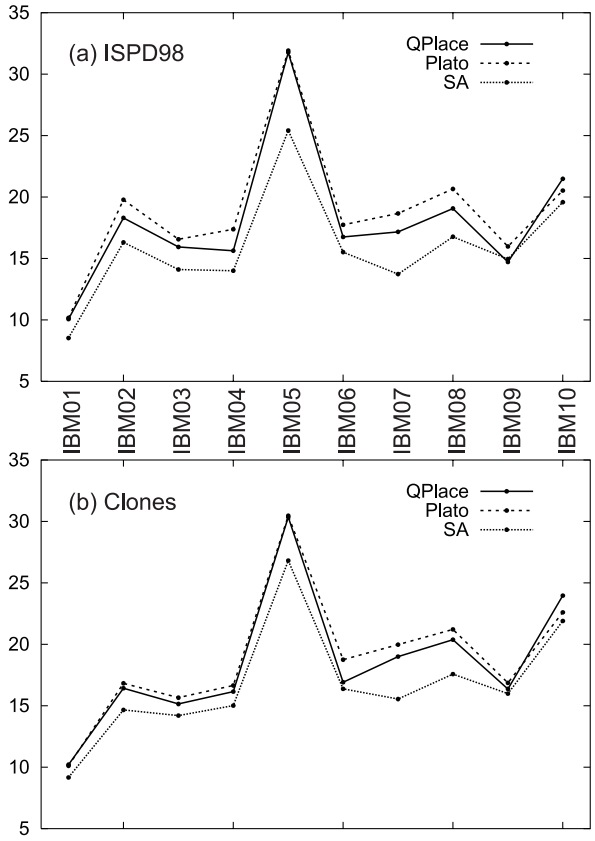

Figure 6: Indirect validation by comparison of different placement algorithms with both (a) real and (b) synthetic benchmark circuits.

size). The important thing to observe is that if one algorithm yields a better average wire length for the original circuit, it should also result in a better average wire length for the synthetic counterpart. The similarity we observe in the trends for both real and synthetic benchmark circuits indicate that the synthetic circuits have a realistic interconnection structure and can be applied for physical design applications.

5 Conclusions

We extended an existing netlist generation algorithm based on bottom-up clustering of subcircuits according to Rent's rule. The stochastic properties of the Rent characteristics are taken into account. As a result the circuits have a very realistic interconnection topology. The structure and complexity of this topology can be fully controlled by generating circuits with different Rent regions.

To make the circuits suitable for timing-driven applications, the combinational loop prevention scheme had to be extended with a delay control mechanism. This mechanism ensures that the length of the combinational paths does not exceed a specified value. The delay distribution can be controlled indirectly by adjusting the target maximum path length distribution of the individual connections.

Direct validation of the synthetic benchmark generation approach by cloning a subset of the ISPD98 benchmark suite and placing these circuits with a commercial placement tool indicates that these synthetic circuits are viable alternatives for real circuits for the purpose of physical design applications. This was confirmed by a more thorough indirect validation based on the comparison of different placement algorithms with both real and synthetic benchmark suites.

References

[1] Computer-Aided Design Benchmarking Laboratory. Available at: http://www.cbl.ncsu.edu/ benchmarks/.

[2] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. of the 1998 Intl. Symp. on Physical Design*, pages 80–85. ACM Press, April 1998.

[3] C. J. Alpert, A. E. Caldwell, A.B. Kahng, and I.L. Markov. Partitioning with terminals: A "new" problem and new benchmarks. In *Proc. of the 1999 Intl. Symp. on Physical Design*, pages 151–157. ACM Press, April 1999.

[4] B. Krishnamurthy. Constructing test cases for partitioning heuristics. *IEEE Trans. on Computers*, C-36(9):1112–1114, September 1987.

[5] P. Verplaetse, J. Van Campenhout, and D. Stroobandt. On synthetic benchmark generation methods. In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pages IV–213–IV–216, May 2000.

[6] D. Stroobandt, P. Verplaetse, and J. Van Campenhout. Generating synthetic benchmark circuits for evaluating CAD tools. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems.*, 19(9):1011–1022, September 2000.

[7] P. Verplaetse, D. Stroobandt, and J. Van Campenhout. A stochastic model for interconnection complexity based on Rent's rule. In *Proc. Intl. Workshop on Logic Synthesis*, pages 319–325, May 2000.

[8] P. Verplaetse, D. Stroobandt, and J. Van Campenhout. A stochastic model for the interconnection topology of digital circuits. *IEEE Trans. on VLSI Systems*, 9(6):938–942, December 2001.

[9] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Comput.*, C–20:1469–1479, 1971.

[10] D. Stroobandt. Analytical methods for a priori wire length estimates in computer systems, November 1998. Ph.D. thesis (translated from Dutch), University of Ghent, Faculty of Applied Sciences.

[11] J. Darnauer and W.W. Dai. A method for generating random circuits and its application to routability measurement. In *Proc. 1996 ACM/SIGDA Intl. Symp. on Field Programmable Gate Arrays*, pages 66–72, February 1996.

[12] G. Karypis and V. Kumar. *hMetis: A Hypergraph Partitioning*
Package, November 1998. Available at: http://www-users.cs.umn.edu/~karypis/metis/hmetis/main.shtml.