Introduction

The circuit Rent parameter has been studied for years in the literature [1, 2]. We revisit the Rent parameter calculation task and provide a program to evaluate the Rent parameter of a given circuit design using different methods, including netlist based methods (e.g., circuit partitioning based methods and graph traversal based methods) and placement based methods (e.g., a rectangle sampling based method). For each evaluation method, different types of pin counting and averaging (e.g., arithmetic and geometric mean) methods are adopted to compute potential values of ‘the’ Rent parameter. The goal of the program is to evaluate the Rent parameter of current circuit designs and analyze the effects of different evaluation methods on the computed Rent parameter. Further, the computed Rent parameter values can be reconciled against extracted wirelength and fanout distributions, in order to inform better predictive models of wiring, buffering and timing in next-generation designs.

Different Pin Counting Methods

In computing the Rent parameter, we first clarify the different pin counting methods. We use the following three different types of pin counting methods based on whether pin directions are differentiated and whether net edges are counted according to a hyperedge view or a graph edge view.

Definition 1. Given a sample closed boundary, Type I pin counting method (graph edge model with differentiated pin directions) counts each source-sink connection crossing the boundary as one pin; Type II pin counting method (hyperedge model with/without differentiated pin directions) counts all source-sink connections of the same net crossing the boundary as one pin; Type III pin counting method (graph edge model without pin directions) counts each connection crossing the boundary as one pin.

Figures 1–4 illustrate the three different pin counting methods defined above. Given a net with 5 pins shown in Figure 1, the number of pins crossing the sample boundary is different for different pin counting methods, e.g., Type I pin counting method yields 2 or 3 for different net topologies as in Figure 2, Type II pin counting method gives 1 regardless of the different net topologies as in Figure 3, and Type III pin counting method...
outputs 6 as in Figure 4. From the above example, we can see that different pin counting methods can result in different computed Rent parameters.

**Different Rent Parameter Evaluation Methods**

The following methods are used to obtain the sample data points for evaluating the different Rent parameters.

**Netlist-based Rent Parameter**

1. Circuit partitioning based method: recursively partition the netlist into smaller partitions using min-cut bisection until the minimum number of cell instances over all
partitions reaches 2. For each level of the recursive bipartitioning, compute the arithmetic/geometric mean values of cell instances and pins, which represent one data point in the fitted curve of the Rent parameter. The classic multilevel circuit partitioner MLPart [3] is incorporated in the program to recursively partition the circuit netlist.

2. **Graph traversal based methods I and II:** start from \( n \) randomly selected cell instances in uniform distribution, perform two min-perimeter clustering algorithms to generate clusters of sizes evenly distributed between 2 and the total number of cell instances. The first min-perimeter clustering algorithm is based on breadth first search (BFS) technique and will not generate disconnected clusters unless necessary; the second min-perimeter clustering algorithm is based priority queue and allows disconnected clusters to be generated whenever there is a smaller cost. For each size of the cluster, compute the arithmetic/geometric mean values of the number of pins, which corresponds to a data point in the fitted curve.

For each of the above Rent parameter evaluation methods, the three different types of pin counting methods are used to compute the Rent parameters.

**Placement-based Rent Parameter**

Given placed cell instances and the corresponding netlist, we use rectangle sampling based methods to compute the placement-based Rent parameters. We randomly sample rectangles in the chip with the centers of the rectangles in uniform distribution. The sizes of the rectangles also uniformly distribute between the size of 2 cell instances and that of the chip. Assume the arithmetic average area of one cell instance is \( A \). And assume the aspect ratio (i.e., \( W_{\text{chip}}/H_{\text{chip}} \)) of the chip is \( A_r \). The width \( W_{\text{min}} \) and height \( H_{\text{min}} \) of the minimum rectangle, whose size is 2 cell instances, are computed as follows.

\[
\begin{align*}
H_{\text{min}} &= \sqrt{2A/A_r} \\
W_{\text{min}} &= H_{\text{min}} \times A_r
\end{align*}
\]

(1) \hspace{1cm} (2)

The size of the sampling rectangles increases with the same aspect ratio \( A_r \) from \( W_{\text{min}} \) and \( H_{\text{min}} \), with fixed \( \delta W \) and \( \delta H \) (\( \delta W/\delta H = A_r \)), to the size of the chip. When the size of a sampling rectangle is small, it may not cover any cell instances. In that case, the sampling rectangle will be discarded and another sampling rectangle will be generated.

1. **Rectangle sampling based method I:** for each rectangle size, take \( n \) samples in the chip, and compute the arithmetic/geometric mean number of cell instances and pins, which yields one data point in the fitted curve.

2. **Rectangle sampling based method II:** generate \( n \) samples for each size of rectangle. For each number of the cell instances in the samples, compute the arithmetic/geometric mean number of pins, which is one data point in the fitted curve.
For each of the above Rent parameter evaluation methods, again, the different types of
the pin counting methods are used separately to compute the Rent parameters.

Table 1 summarizes the total 30 different Rent parameters that the program will compute
for each testcase combining the different evaluation methods discussed above.

<table>
<thead>
<tr>
<th>Evaluation method</th>
<th>Pin counting method</th>
<th>Averaging method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit partitioning based method</td>
<td>Type I</td>
<td>arithmetic mean</td>
</tr>
<tr>
<td>Graph traversal based method I</td>
<td>Type II</td>
<td>geometric mean</td>
</tr>
<tr>
<td>Graph traversal based method II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rectangle sampling based method I</td>
<td>Type III</td>
<td></td>
</tr>
<tr>
<td>Rectangle sampling based method II</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Region I Rent Parameter Determination**

Given a set of sampling points corresponding to the pairs of the gate number \( N \) and the
terminal number \( T \) of each module, we compute the Rent parameter \( p \) in \( T = k \times N^p \) by
linear regression, i.e., \( \log(T) = \log(k) + p \times \log(N) \), where \( k \) is the average number of pins
for each gate. To determine Region I, we iteratively remove the sampling point with the
maximum number of gates in a module, and compute the new Rent parameter \( p \) and the
standard deviation \( \sigma \) of the sampling points over the fitted line. The sampling point
removal process continues until \( \sigma \) cannot be improved or the total number of sampling
points reaches \( \frac{3}{4} \) of the original.

**Installation and Testing of the Program**

The current version of the program is tested on Red Hat Linux g++ (GCC) 3.2.3 or higher.
To install the program, please follow the following steps.

1. Clean the existing objective files and libraries:
   ```
   csh>  cd ./WLD-V1/RentCon/Source
   csh>  make -f makefile.install clean
   ```

2. Configure, compile and generate the executable:
   ```
   csh>  ./configure
   ```
To test the program, please invoke the executable with the following options. Note that backslash (\") can be used for continuation in providing the parameters in multiple lines.

Options:

- `-f 'aux file name'`: auxiliary file name to read in. All DEF/LEF files and the unused cell master names are given in the auxiliary file. Both relative and absolute paths are supported for LEF/DEF file names. Each line in the auxiliary file cannot exceed 5000 characters. Backslash (\") at the end of a line (except keywords) can be used for continuation with the following line, so that a LEF/DEF file or cell master name can span multiple lines. The cell master names are given to specify a set of cell instances excluded in Rent’s parameter computation, e.g., filler cells and decaps should not be used for computing Rent’s parameter. Two methods are given for specifying the cell master names, i.e., matching by key word and matching by full name. “Matching by key word” is denoted by “REMOVE_CELL_BY_KEY”, where each cell master name with a substring as one of the given keywords needs to be removed; and “matching by full name” is denoted by “REMOVE_CELL_BY_FULL_NAME”, where each cell master name exactly matching one of the given cell names needs to be removed during the computation. There is a sample file “Rent.aux” accompanying the source code. The grammar of the auxiliary file is as follows.

  - Lines starting with “#” are comments
  - Key words: DEF (def), LEF (lef), REMOVE_CELL_BY_KEY,
    REMOVE_CELL_BY_FULL_NAME
  - `<aux file> ::= <DEF statement> <newline> <LEF statement> <newline>
    <Cell removal statement> | <LEF statement> <newline> <DEF statement>
    <newline> <Cell removal statement>
  - `<DEF statement> ::= DEF : <separator> <name list>
  - `<LEF statement> ::= LEF : <separator> <name list>
  - `<Cell removal statement> ::= <newline> | <Cell key removal statement> |
    <Cell full name removal statement> | <Cell key removal statement>
  - `<newline> ::= <\n> | <\n> <newline>
  - `<separator> ::= <\t> | <' '> | <\t> <separator> | <' '> <separator>
  - `<name list> ::= <file name> | <file name><separator><name list>
  - `<Cell key removal statement> ::= REMOVE_CELL_BY_KEY :
    <separator> <name list>
  - `<Cell full name removal statement> ::= REMOVE_CELL_BY_FULL_NAME :
    <separator> <name list>
  - `<newline> ::= <\n> | <\n> <newline>
  - `<separator> ::= <\t> | <' '> | <\t> <separator> | <' '> <separator>

- `-verb 'i,j,k'`: i, j and k are unsigned integer values, where i is the verbosity for actions, i.e., writing “doing this, doing that” in more or less detail, depending on the level, j is the verbosity for system resources, i.e., writing how much
memory/CPU time/etc was used, in more or fewer places, depending on the level, and \( k \) is the verbosity for major stats, i.e., writing quantities/sizes of important components, on more or fewer occasions, depending on the level. Level 0 means nothing will be written to the standard output. Currently the max level supported is 3.

- \(-\log\) 'file name': print all output (including standard output and standard error) to a file instead of the screen

- \(-\text{noCP}\): do not use circuit partitioning method. The method is used by default.

- \(-\text{noGT}\): do not use graph traversal method. The method is used by default.

- \(-\text{noRS}\): do not use rectangle sampling method. The method is used by default.

- \(-\text{seedCP}\) 'seed': unsigned integer value ‘seed’ for the random number generator used in circuit partitioning method. Default value is 123.

- \(-\text{seedGT}\) 'seed': unsigned integer value ‘seed’ for the random number generator used in graph traversal method. Default value is 123.

- \(-\text{seedRS}\) 'seed': unsigned integer value ‘seed’ for the random number generator used in rectangle sampling method. Default value is 123.

- \(-\text{startsPerRun}\) 'startsPerRun': number of solutions for each run of circuit partitioning process. The circuit partitioning program chooses the best partitioning solution among all the solutions obtained. Default value is 1. A larger value can result in improved solution quality at the cost of higher runtime.

- \(-\text{totalRuns}\) 'totalRuns': total number of runs of circuit partitioning process. The partitioning program chooses the best solution obtained from all the runs. Default value is 1. A larger value can result in improved partitioning solution quality (and lower Rent parameter) at the cost of higher runtime.

- \(-\text{sampleNumGT}\) 'sampleNum': number of samples computed in the graph traversal method. Default value is 3.

- \(-\text{sampleNumRS}\) 'sampleNum': number of samples computed in the rectangle sampling method. Default value is 3.

- \(-\text{sampleSizeNum}\) 'sampleSizeNum': total number of rounds in the graph traversal and rectangle sampling methods, i.e., the granularities used in the graph traversal and rectangle sampling based evaluation methods. Default value is 30. A larger value corresponds to a finer granularity at the cost of higher runtime.
• **-h | -help:** print the simplified description of the above options.

Example: If your design is composed of one DEF (top.def) file and three LEF (tech.lef, std.lef, mem.lef) files in different directories, and cell master names with substrings “FILLER” or “DECAP”, or exactly matching “FILLX2” or “FILLX4” need to be removed, a sample auxiliary file is as follows (backslash ‘\’ is used for continuation)

```plaintext
#auxiliary file Rent.aux
DEF:
../relativepath\
/top.def

LEF:
/absolutepath1/\
techn.lef
/absolutepath2/\
/std.lef . /mem.lef

#FILLER and DECAP are substrings of the cell masters
REMOVE_CELL_BY_KEY:
FILLER
DECAP

#FILLX2 and FILLX4 are exact cell master names
REMOVE_CELL_BY_FULL_NAME:
FILLX2
FILLX4

```

csh> RentCon.exe -f Rent.aux -verb 1_1_1 -seedCP 100 \ 
-seedGT 100 -seedRS 100 -startsPerRun 1 \ 
-totalRuns 1 -sampleNumGT 3 -sampleNumRS 3 \ 
-sampleSizeNum 15 #optionally with -log RentCon.log

You can try the program on the given small example design.

```bash
  csh> cd ./WLD-V1/TestCase/sample
csh> source run-rent.csh
```

After running the script, you can find an output file and a log file in the working directory.

“RentCon.log” and “RentParam.txt”

Please use options “--verb 1_1_1 --log RentCon.log” to test the program and send the output files on your designs to UCSD. If there is a problem during execution of the program, please use options “--verb 3_3_3 --log RentCon.log” to test the program and send the log file for diagnosing.

Kwangok Jeong (kjeong@vlsicad.ucsd.edu)
Andrew B. Kahng (abk@cs.ucsd.edu)
Hailong Yao (hailong@cs.ucsd.edu)
NOTE: The program does not extract any technology information or logical information of the design. Output files are in ASCII format, so that you can check the contents.

References


APPENDIX

Testing on the Sensitivity of Results to Various Options

To test the sensitivity of the results to various options, the following settings of the command-line parameters are recommended.

For circuit partitioning method, there are 8 cases with different seeds as follows (4 cases with the same seed).
1. -seedCP 500, 1000
2. -startsPerRun 2 5
3. -totalRuns 2 5

For graph traversal method, there are 8 cases with different seeds as follows (4 cases with the same seed)

-seedGT 500, 1000
-sampleNumGT 10 20
-sampleSizeNum 20 50

For rectangle sampling method, there are 8 cases with different seeds as follows (4 cases with the same seed)

-seedRS 500, 1000
-sampleNumRS 10, 20
-sampleSizeNum 20, 50

In total, there are total 24 cases with different seeds (or 12 cases using the same seed) that can be used to test the sensitivity of the results to the various options.