On The Circuit Implementation Problem

WingNing Li† Andrew Lim‡ Prathima Agrawal§ Sartaj Sahni‡
† Dept. of Computer Science, University of Arkansas
‡ Information Technology Institute, Republic of Singapore
§ Dept. of Computer and Info. Sciences, University of Florida
‡ AT&T Bell Lab., Murray Hill, New Jersey

Abstract

We consider the problem of selecting an implementation of each circuit module from a cell library so as to satisfy overall delay and area (or delay and power) requirements. Two versions of the circuit implementation problem, basic circuit implementation problem and the general circuit implementation problem are shown to be NP-hard. A pseudo-polynomial time algorithm for the basic circuit implementation problem on series-parallel circuits is developed, and heuristics for the basic circuit implementation problem on general circuits are formulated and experimented with.

1 Introduction

In this paper we examine the problem of selecting, from a precharacterized library, an implementation of each module in a combinatorial logic circuit such that the resulting circuit implementation satisfies the timing and area (power) requirements. While our development applies equally to the cases of satisfying timing and area requirements and satisfying timing and power requirements, in what follows, we explicitly refer only to the case of satisfying timing and area requirements.

We define the circuit implementation problem as that of selecting an implementation for each of the circuit modules so that the area and delay of the overall circuit do not exceed some specified limits $\alpha$ and $\delta$ respectively. In the basic circuit implementation problem (BCI), the delay between every pair of input-output terminals of a module is the same. In the general circuit implementation problem (GCI), the delay between different pairs of input-output terminals of a module maybe different. The former model is more applicable for simple gates and the latter for compound gates.

In this paper, we show, in Section 3, that the basic circuit implementation problem is NP-hard even when the circuits are chains of components. In the same section we show that the general circuit implementation problem is NP-hard even if all implementations of all modules have the same area. By using suitable decomposition techniques and dynamic programming, we are able to develop (Section 4) a pseudo-polynomial time algorithm that obtains optimal solutions for basic series-parallel circuits. This result contrasts with that of McNall and Caasman[1] who have developed a similar dynamic programming algorithm for the synthesis of pipelined architectures.

In Section 5, we propose several heuristics for basic circuits. These heuristics of Section 5 are evaluated experimentally using ISCAS benchmark circuits. The results of this evaluation are presented in Section 6.

Chan[2] has studied a circuit implementation problem that is closely related to ours. In his version of the problem, there is a minimum and maximum delay requirement for the realized circuit (rather than just a maximum delay). He has obtained a pseudo-polynomial time algorithm to obtain a minimum area realization of a tree. The obtained realization satisfies the delay constraints. Our pseudo-polynomial time algorithm is easily generalized to handle the additional delay constraint and applies not only to trees but also to series-parallel graphs. Chan[2] also proposes a backtracking algorithm for circuit modeled by dags. While this algorithm obtains circuit realizations that satisfy the delay constraint (when such a realization exist), it does not guarantee optimal area realizations. In addition to generating suboptimal circuit implementations, this backtracking algorithm is of exponential complexity and so is applicable only to relative small circuits. Chan[2] also shown that the circuit realization problem with minimum and maximum delay constraints is NP-hard for basic circuits with tree topologies. Our NP-hard result for chains is a stronger result as a chain is a special case of a tree.

The circuit implementation problem studied here is also related to the technology mapping problem studied earlier by Darringer et al.[3], Gregory et al.[4], Keutzer[5], Detjens et al.[6], Touati[7], and others. Because of space limitations, we shall not elaborate on this relationship.
A circuit is an interconnected set of modules. A \( p \) input \( q \) output module has \( p \) input pins and \( q \) output pins. Such a module is called a \((p,q)\)-module. The interconnects of the circuit connect an output pin of one module to an input pin of another module. Some input pins are labeled as primary inputs and some output pins are labeled as primary outputs. We shall limit ourselves to circuits in which the modules can be ordered such that every interconnect connects an output pin of one module to an input pin of another module to its right. So, all interconnects are directed left to right. The graph for such a circuit is directed and acyclic, i.e., it is a dag.

Suppose there are \( n \), possible implementations of module \( i \) and that module \( i \) is a \((p_i,q_i)\)-module. Let \( A(i,j) \) denote the area of the \( j \)th implementation of this module, \( 1 \leq j \leq n \), and let \( D(i,j,u,v) \) be the delay from input \( u \) to output \( v \) of the \( j \)th implementation of module \( i, 1 \leq j \leq n, 1 \leq u \leq p_i, 1 \leq v \leq q_i \). For basic circuits, \( D(i,j,u,v) = d_{ij}, 1 \leq u \leq p_i, 1 \leq v \leq q_i \). The area of the implementation \( S = (s_1,s_2,\ldots,s_n) \) is \( \sum_{i=1}^{n} A(i,s_i) \).

### 3 Complexity Results

We first define the following known NP-complete problem \cite{Garey90}:

**PARTITION**

Input: A finite set \( B \) and a size \( s(b) \in \mathbb{Z}^+ \), for each \( b \in B \).

Output: "Yes" if there is a subset \( B' \subset B \) such that \( \sum_{b \in B} s(b) = \sum_{b \in B-B'} s(b) \).

**Theorem 1** The BCI problem is NP-hard even when all modules are \((1,1)\)-modules interconnected to form a chain and each module has only two possible implementations.

**Proof** We shall show that any instance of the partition problem can be transformed, in polynomial time, into an instance of the BCI problem for which there is an \((\alpha, \delta)\) implementation iff the output to the partition instance is "yes". The constructed BCI instance satisfies the remaining requirements of the theorem. The BCI instance is obtained from the partition instance as follows. For each \( b \in B \), we have a distinct \((1,1)\)-module, \( i, \) with two possible implementations.

\[
A(i,1) = s(b), \quad D(i,1,1,1) = 0
\]

\[
A(i,2) = 0, \quad D(i,2,1,1) = s(b)
\]

The modules are connected in any order to form a chain and the values of \( \alpha \) and \( \delta \) are \( \alpha = \delta = \sum_{b \in B} s(b)/2 \). One may easily verify that the constructed BCI instance has an \((\alpha, \delta)\) implementation iff the corresponding partition instance has output "yes". \( \square \)

**Theorem 2** The GCI problem is NP-hard even if all implementations of all modules have the same area.

**Proof** See \cite{Garey90}. \( \square \)

### 4 Series-Parallel Circuits

A simple parallel circuit is a basic circuit that is comprised of several chains that have the same first and last module. A series-parallel circuit is a basic circuit recursively defined as:

- A chain of basic modules is a series-parallel circuit
- A simple parallel circuit is a series-parallel circuit
- A circuit obtained from a series-parallel circuit \( C \) by replacing any interconnect of \( C \) by another series-parallel circuit is also a series-parallel circuit.

We shall assume that the area and delay values associated with each implementation of each module are integers. An algorithm for the basic circuit realization problem is a pseudo-polynomial time algorithm if its complexity is some polynomial in \( n \) (the number of modules), \( m \) (maximum possible implementations of any module), area (the maximum allowable area), and delay (the maximum allowable delay).

An implementation that has area \( a \) and delay \( d \) is an optimal implementation iff there is no implementation with area \( \leq a \) and delay \( < d \) or with area \( < a \) and delay \( \leq d \).

#### 4.1 Chains

The optimal implementations for a chain may be obtained by enumerating the different possibilities using a dynamic programming type approach (Chapter 5, \cite{Garey90}). Suppose the chain has \( c \) modules \( M_1, M_2, \ldots, M_c \) where \( M_1 \) is to the left of \( M_{i+1} \), \( 1 \leq i < c \). Let \( L_r \) be an ordered list of pairs \((a_i,d_i)\) where \( a_i \) and \( d_i \) are, respectively, the area and delay associated with an optimal implementation of the chain \( M_1, M_2, \ldots, M_r \). Let \( L_r \) have the form:

\[
L_r = \{(a_1,d_1),\ldots,(a_k,d_k)\} \quad \text{where} \quad a_i < a_{i+1} \quad \text{and} \quad d_i > d_{i+1}, 1 \leq i < k
\]

\( L_r \) cannot have two pairs \((a_i,d_i)\) and \((a_j,d_j)\) such that \((a_i \leq a_j \text{ and } d_i < d_j)\) or \((a_i < a_j \text{ and } d_i \leq d_j)\).

The set \( L_1 \) is simply those implementations of \( M_1 \) that are optimal. Hence if the set of implementations of \( M_1 \) includes pairs \((a_i,d_i)\) and \((a_j,d_j)\) that satisfy (2) then the pair \((a_i,d_i)\) is eliminated. Note that a suboptimal implementation of \( M_1 \) cannot be used in any optimal implementation of any of the chains \( M_1, M_2, \ldots, M_s \), where \( 1 \leq s \leq c \). Similarly no suboptimal implementation of the chain \( M_1, M_2, \ldots, M_s \) can be used to obtain an optimal implementation of \( M_1, M_2, \ldots, M_s, 1 \leq s < c \). From \( L_r \) we may obtain \( L_{r+1} \) by considering each of the possible optimal implementations of \( M_{r+1} \). For each optimal implementation \((a_i,d_i)\) of \( M_{r+1} \) we create an ordered list \( L_{r+1} \) as below:

\[
L_{r+1} = \emptyset
\]

for \( i = 1 \) to \(|L_r|\) do

append \((a_i+a_s,d_i+d_s)\) to \( L_{r+1} \);

\( \{(a_i,d_i)\text{ is the } i\text{th pair in } L_r\} \)

end.
Next, the lists \( L_{r+1} \), \( 1 \leq r \leq \text{number of optimal implementations of} \( M_{r+1} \) are merged to obtain \( L_{r+1} \). The merging is done in a way that \( L_{r+1} \) satisfies (1). This requires the elimination of pairs \( (a_i, d_i) \) that satisfy (2).

Using this strategy we begin with \( L_1 \), then obtain \( L_2 \), then \( L_3 \), ..., and finally \( L_r \). For each pair \( (a_i, d_i) \) in \( L_r \) the actual implementation can be obtained using a traceback as in Section 5.5 of [10].

The above algorithm for a chain also allows us to obtain a chain to module transformation in which any chain of modules can be replaced by a single module, the delay and area of whose possible implementations are given by the pairs in \( L_c \).

4.2 Simple Parallel Circuits

A basic tree circuit has two columns of basic modules as in Figure 1(a). The second column has exactly one module. The area and delay pairs of the optimal implementations of a basic tree circuit can be obtained by first obtaining those for the column one modules. This allows us to replace the circuit of Figure 1(a) with the two module chain of Figure 1(b) where the possible implementations of module A have area and delay corresponding to the optimal implementations of the column 1 modules. The optimal implementation of the two module chain of Figure 1(b) may be obtained using the algorithm for a chain. In this way, the basic tree of Figure 1(a) is transformed into a single module as in Figure 1(c).

By beginning at the leaves and using this transformation repeatedly, we can obtain the optimal implementations of any tree.

The \( c \) modules of column 1 of the basic tree of Figure 1(a) may be transformed into the equivalent single module A of Figure 1(b) by constructing the ordered lists \( L_1, L_2, \ldots, L_c \) as was done for a chain. In the code to construct \( L_{r+1} \), we replace \( (a_i + a_r, d_i + d_r) \) by \( (a_i + a_r, \max \{d_i, d_r\}) \).

The optimal implementations of a simple parallel graph can be obtained by using the transformations shown in Figure 2. The steps are:

**Step 1:**
The modules from \( s \) to \( f \), of Figure 2(a) form a chain. Using the chain transformation of Section 4.1 these are replaced by an equivalent single module \( A \).

**Step 2:**
The modules \( A_1, A_2, \ldots, A_c \) of Figure 2(b) are replaced by a single module \( D \) using the transformation used to go from Figure 1(a) to Figure 1(b).

**Step 3:**
Use the chain transformation of Section 4.1 to go from Figure 2(c) to Figure 2(d).

4.3 Series-Parallel Circuits

The optimal implementation of any series-parallel circuit can be obtained by repeatedly using the chain and simple parallel circuit transformations on sub-circuits of the given series-parallel circuit. The time complexity of the algorithm remains pseudo-polynomial.

In case the series-parallel decomposition of the circuit isn't already known, it can be obtained in linear time using the algorithm of Valdes[11].

5 Heuristics For Basic Circuits modeled by DAGs

In this section, we develop six heuristics to obtain minimal area circuit implementations given a delay constraint \( \delta \). All are for the basic circuit implementation problem.

Common to all of the heuristics is a preprocessing step in which the following transformations are made:

**T1:** [Elimination of module chains]
Each chain of modules is replaced by a single module whose possible implementations are the optimal implementations of the chain. This is done using the algorithm in section 4.1.

**T2:** [Elimination of redundant edges]
An edge \( <u,v> \) of the circuit graph is redundant if the graph contains a \( u \) to \( v \) path that does not use this edge. Since the delay on such a path is no less than the delay on the edge \( <u,v> \), the edge \( <u,v> \) may be removed from the circuit. All redundant edges are removed in this transformation.

**T3:** [Elimination of suboptimal implementations]
Let \( (a_i, d_i) \) and \( (a_j, d_j) \) be the areas and delays of two different implementations of the same module. If \( a_i \leq a_j \) and \( d_i \leq d_j \), then the implementation \( (a_j, d_j) \) is suboptimal and is eliminated.
Step 1: Select an initial implementation for each module.
Step 2: Consider the modules, one at a time, in some order and revise the selected implementation of each.
Step 3: This is a postprocessing step used by only one of our heuristics to further revise the selected implementation of each module.

The general structure of our heuristics is described in Figure 3. The details of each are provided in the next six subsections.

5.1 Random Heuristic
In this, the initial module implementations of step 1 are selected at random and the implementations revised in step 2 by considering the modules in some random order. When a module is considered for revision in step 2, an implementation with the least area that causes this module to be on no path with length $> \delta$ is chosen. If no such implementation exists, then the least delay (i.e., fastest) implementation for this module is selected. For this heuristic, step 3 is null. This heuristic is easily implemented to run in $O((|V|(|V| + |E|))$ time where $|V|$ is the number of vertices in the circuit graph and $|E|$ is the number of edges in this graph. When a module is being considered in step 2, we need to know the length $l$ of the longest delay path it is on. Let the delay of this module’s current implementation be $d_{\text{current}}$. Then for this module to be on no path with delay $> \delta$, its implementation should have delay $\leq \delta - l + d_{\text{current}}$.

5.2 Topological Heuristic
In step 1, the fastest implementation for each heuristic is selected. In step 2, the modules are considered in two different orders: topological and reverse topological. The better of the two resultant circuit implementations is used. When a module is considered in step 2, the implementation with the least area that results in the maximum area reduction. For the initial implementation of step 1, this heuristic uses the fastest implementation of each module. In step 2, the order in which the modules are considered is obtained dynamically. The next module to consider is one which results in the maximum reduction in the circuit area. For this, each of the modules not yet considered is evaluated to determine the area reduction possible by changing its selected implementation. If the delay of the longest path through the module is $d$, then we can choose an implementation with delay upto $\delta - d$ larger than the delay of its currently selected implementation. If the possible implementations are ordered by delay (and hence by area), a binary search can be performed to obtained the implementation that results in the maximum area reduction.

As for the other heuristics described so far, step 3 of this heuristic is also null. The complexity of the greedy heuristic is $O((|V|(|V| \log m + |E|))$ where $m$ is the maximum number of possible implementations for any module.

5.3 Static Delay Heuristic
The initial step 1 implementations are the slowest ones for each module. To obtain the order in which modules are considered in step 2, we compute the number of input to output paths that pass through each module. This can be done in $O(|V| + |E|)$ time by considering the modules in topological and reverse topological order. In step 2, the modules are considered in nonincreasing order of the number of input to output paths through them. The implementation revision scheme is the same as that for the random heuristic.

5.4 Greedy Heuristic
For the initial implementation of step 1, this heuristic uses the fastest implementation of each module. In step 2, the order in which the modules are considered is obtained dynamically. The next module to consider is one which results in the maximum reduction in the circuit area. For this, each of the modules not yet considered is evaluated to determine the area reduction possible by changing its selected implementation. If the delay of the longest path through the module is $d$, then we can choose an implementation with delay upto $\delta - d$ larger than the delay of its currently selected implementation. If the possible implementations are ordered by delay (and hence by area), a binary search can be performed to obtained the implementation that results in the maximum area reduction.

As for the other heuristics described so far, step 3 of this heuristic is also null. The complexity of the greedy heuristic is $O((|V|(|V| \log m + |E|))$ where $m$ is the maximum number of possible implementations for any module.

5.5 Static Area Heuristic
The fastest implementation of each module is used in step 1, as the initial implementation. In step 2, the modules are considered in nondecreasing order of the number of input to output paths through them. When module $i$ is being considered in step 2, each of its possible implementations is given a score and the implementation with highest score is selected. The score of implementation $j$ of module $i$ is zero if using implementation $j$ results in a circuit delay $> \delta$. Otherwise, its score is obtained using the formula:

$$\text{score}(i, j) = \Delta a_i (\alpha + \beta f(i, j) + \gamma)$$

where $\Delta a_i$ is the area reduction that results from using implementation $j$ over the currently selected implementation, $\alpha$, $\beta$, and $\gamma$ are constants that were set to 4, 1, and 1 respectively. These values for $\alpha$, $\beta$, and $\gamma$ were determined experimentally using half of our test circuits. While the performance of the heuristic is sensitive to the choice of these constants, the chosen values tended to produce uniformly good solutions over the range of circuits tested. The function $f$ measures the undesirability of implementation $j$ of module $i$, $f$ is given by:

$$f(i, j) = \sum_{<k,i> \in E} \max[0, \text{right}(i) + d_{ij} - \text{right}(k)] \times LP(k)$$

$$+ \sum_{<k,i> \in E} \max[0, \text{left}(i) + d_{ij} - \text{left}(k)] \times RP(k)$$

where $\text{right}(i)$ is the length of the longest delay path from module $i$ to a primary output, this length does not include the delay of module $i$; $\text{left}(i)$ is the length of the longest delay path from a primary input to module $i$; $d_{ij}$ is the delay of the $j$th implementation of module $i$; $LP(k)$ is the number of paths from the primary inputs to module $k$; $RP(k)$ is the number of paths from the module $k$ to primary outputs;
The asymptotic complexity of this heuristic is $O(|V|(|V|m + |E|))$.

6 Experimental Results

Our algorithms were programmed in C and the programs run on a SUN SPARCstation 1. For test data, we used the ten ISCAS circuits described in Brglez [12] and another circuit (C17) described in Chan [2] together with our precharacterized cell library.

The run times of the six heuristics are tabulated in the last six columns of Table 1. The times are in seconds and represent the average time needed to find a solution for each of the 6 values within the range in column 5. Table 2 provides the average area obtained for each $\delta$ value and the percentage of improvement when compared to the random heuristic. The dynamic area heuristic was able to reduce area by as much as 60% on some of the test circuits and for some $\delta$ values. The average area reduction was as much as 35% on some circuits. The dynamic area heuristic obtained optimal solutions for all but one of the $\delta$ values for that circuit.

From the experiments, we made several observations:

- All the heuristics are fast and can easily handle large circuits.
- The run time of the heuristics is highly dependent on the depth of the circuit. This is due to the amount of time required to update the circuit once a module changes its delay and area.
- Simple methods like the first four heuristics are not as effective as dynamic area heuristics like the static area and dynamic area heuristics.
- Starting with the minimum area circuit with a large delay and then trying to reduce the delay of paths that violate timing requirement requires more sophisticated techniques as the delay of a circuit is the maximum delay of all paths in the circuit whereas the area of the circuit is the summation of all the modules in the circuit.

7 Conclusion

We have shown that the basic circuit implementation problem and the general circuit implementation problem are NP-hard. We developed a pseudo-polynomial time algorithm for the basic circuit implementation problem on series-parallel circuits. Our algorithm can easily be generalized to circuits in which a minimum and maximum source to module delay is associated with each module. This includes the case when a minimum and maximum delay for the entire circuit is provided. Thus the results of Chan [2] for trees are subsumed by our algorithm. Furthermore, we develop 2 fast and effective heuristics (static area and dynamic area) for the basic circuit implementation problem and compare them exhaustively with 4 other reasonable approaches. Experimental results confirmed the superiority of our static area and dynamic area heuristics. While our discussion considered the circuit implementation problem in terms of satisfying timing and
Table 1: Characteristics of the test data and run times

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>No. of Nodes</th>
<th>No. of Edges</th>
<th>Depth of Circuit</th>
<th>Delays Tested</th>
<th>Random Topo</th>
<th>Static D'</th>
<th>Greedy Static A</th>
<th>Dyn. A</th>
</tr>
</thead>
<tbody>
<tr>
<td>C17</td>
<td>13</td>
<td>14</td>
<td>5</td>
<td>19-29</td>
<td>0.003</td>
<td>0.001</td>
<td>0.003</td>
<td>0.001</td>
</tr>
<tr>
<td>C432</td>
<td>350</td>
<td>426</td>
<td>22</td>
<td>22-44</td>
<td>0.08</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>C499</td>
<td>585</td>
<td>928</td>
<td>24</td>
<td>24-48</td>
<td>0.27</td>
<td>0.20</td>
<td>0.22</td>
<td>0.18</td>
</tr>
<tr>
<td>C880</td>
<td>443</td>
<td>759</td>
<td>25</td>
<td>25-50</td>
<td>0.13</td>
<td>0.15</td>
<td>0.11</td>
<td>0.33</td>
</tr>
<tr>
<td>C1355</td>
<td>587</td>
<td>1064</td>
<td>25</td>
<td>25-50</td>
<td>0.24</td>
<td>0.25</td>
<td>0.22</td>
<td>0.19</td>
</tr>
<tr>
<td>C1908</td>
<td>913</td>
<td>1498</td>
<td>41</td>
<td>41-82</td>
<td>0.43</td>
<td>0.59</td>
<td>0.42</td>
<td>1.38</td>
</tr>
<tr>
<td>C2670</td>
<td>1436</td>
<td>2076</td>
<td>33</td>
<td>33-66</td>
<td>0.52</td>
<td>0.69</td>
<td>0.42</td>
<td>2.64</td>
</tr>
<tr>
<td>C3540</td>
<td>1719</td>
<td>2939</td>
<td>48</td>
<td>48-96</td>
<td>0.93</td>
<td>1.53</td>
<td>0.81</td>
<td>4.45</td>
</tr>
<tr>
<td>C5315</td>
<td>2485</td>
<td>4386</td>
<td>50</td>
<td>50-100</td>
<td>1.09</td>
<td>1.55</td>
<td>0.90</td>
<td>4.58</td>
</tr>
<tr>
<td>C6288</td>
<td>2448</td>
<td>4800</td>
<td>125</td>
<td>125-250</td>
<td>4.03</td>
<td>7.11</td>
<td>3.75</td>
<td>18.5</td>
</tr>
<tr>
<td>C7552</td>
<td>3719</td>
<td>6144</td>
<td>44</td>
<td>44-88</td>
<td>1.73</td>
<td>2.59</td>
<td>1.25</td>
<td>20.4</td>
</tr>
</tbody>
</table>

Table 2: Results for Basic Circuit Implementation problem

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Random Topo Order</th>
<th>Static D</th>
<th>Greedy Static A</th>
<th>Dyn. A</th>
</tr>
</thead>
<tbody>
<tr>
<td>C17</td>
<td>26.57</td>
<td>24.45</td>
<td>26.08</td>
<td>28.54</td>
</tr>
<tr>
<td>C432</td>
<td>810</td>
<td>896</td>
<td>26.4</td>
<td>569</td>
</tr>
<tr>
<td>C499</td>
<td>1905</td>
<td>1488</td>
<td>21.9</td>
<td>1595</td>
</tr>
<tr>
<td>C880</td>
<td>1008</td>
<td>901</td>
<td>10.6</td>
<td>905</td>
</tr>
<tr>
<td>C1355</td>
<td>2078</td>
<td>1722</td>
<td>16.6</td>
<td>1803</td>
</tr>
<tr>
<td>C1908</td>
<td>2396</td>
<td>1871</td>
<td>15.0</td>
<td>2034</td>
</tr>
<tr>
<td>C2670</td>
<td>3026</td>
<td>2748</td>
<td>9.19</td>
<td>2511</td>
</tr>
<tr>
<td>C3540</td>
<td>4255</td>
<td>3264</td>
<td>23.3</td>
<td>3508</td>
</tr>
<tr>
<td>C5315</td>
<td>3893</td>
<td>3628</td>
<td>6.6</td>
<td>3543</td>
</tr>
<tr>
<td>C6288</td>
<td>7672</td>
<td>6941</td>
<td>9.53</td>
<td>6396</td>
</tr>
<tr>
<td>C7552</td>
<td>6544</td>
<td>6403</td>
<td>2.18</td>
<td>5947</td>
</tr>
</tbody>
</table>

References


