# Graph Matching-Based Algorithms for FPGA Segmentation Design *†

*Yao-Wen Chang*[1], *Jai-Ming Lin*[1], *and D. F. Wong*[2]

[1]Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan
[2]Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712, USA

## Abstract

*Process technology advances will soon make the one-million gate FPGA a reality. A key issue that needs to be solved for the large-scale FPGAs to realize their full potential lies in the design of their segmentation architectures [10]. One-dimensional segmentation designs have been studied to some degree in much of the literature; most of the previously proposed methods are based on stochastic or analytical analysis. In this paper, we address a new direction for studying segmentation architectures. Our method is based on graph-theoretic formulation. We first formulate a net matching problem and present a polynomial-time optimal algorithm to solve the problem. Based on the solution to the problem, we develop an effective and efficient matching-based algorithm for FPGA segmentation designs. Experimental results show that our method significantly outperforms previous work. For example, our method achieves* averages *of 18.2% and 8.9% improvements in routability, compared with the work in [14] and the most recent work in [7], respectively. More importantly, our approaches are very flexible and can readily extend to higher-order segmentation designs (e.g., two- or three-dimensional segmentation design, etc), which are crucial to the design of large-scale FPGAs.*

## 1 Introduction

With the advances in process technology, one-million gate *Field-Programmable Gate Arrays* (*FPGAs*) will soon become available. A key issue that needs to be solved for the large-scale FPGAs to realize their full potential lies in the design of their routing architectures [10].

### 1.1 FPGA Architectures

Figures 1(a) and (b) show two major types of FPGA architectures: array-based [2, 12] and row-based FPGAs [1, 3]. An array-based FPGA (see Figure 1(a)) is composed of a two-dimensional array of *logic modules* that can be connected with general routing resources. The logic modules (denoted by $L$) are used to implement logic functions. The routing resources comprise vertical and horizontal channels. A cross area of vertical and horizontal channels is referred to as a *switch module* (denoted by $S$). Each side of a switch module is linked with a set of segments. Segments on different sides of a switch module can be connected together through the switch module to form a longer connection. The routing channels in an array-based FPGA usually consist of various types of interconnect, distinguished by the relative length of their segments. For example, the routing channel in a Xilinx XC4000EX-series FPGA contains *single-length lines*, *double-length lines*, *quad-length lines*, and *long-lines* [12]. (Figure 2 shows three types of segments in a routing channel; note that quad-length lines are not shown in the figure.) The single-length lines form a grid of horizontal and vertical lines that intersect at switch modules. The double-length (quad-length) lines consist of a grid of segments twice (quadruple) as long as the single-length lines. The longlines are a grid of segments that run the entire vertical or horizontal channel.

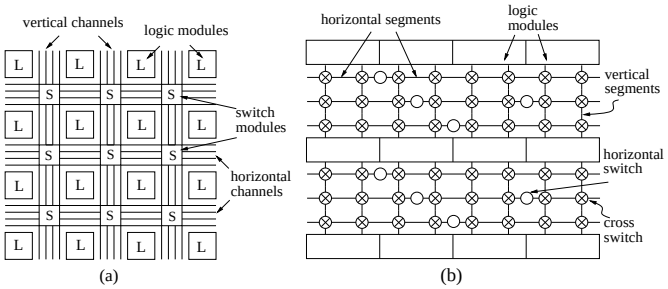


Figure 1: (a) The array-based FPGA architecture. (b) The row-based FPGA architecture.

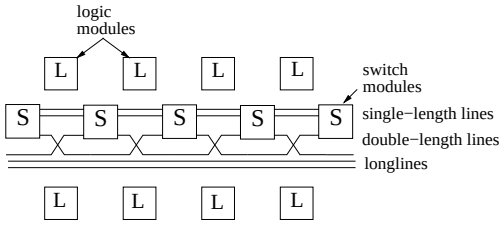


Figure 2: Three types of segments in a routing channel of the array-based FPGA.

The architecture of a row-based FPGA (see Figure 1(b)) is analogous to a standard cell. The logic modules are placed in parallel in predefined locations, and channels are settled between two neighboring rows of logic modules. Each logic module is linked with vertical segments for input and output. A vertical segment can be connected to a horizontal segment by programming a *cross switch* (denoted by ⊗) to be ON. The routing tracks are divided into several segments of different lengths. Two neighboring segments can be connected together to establish a longer connection by programming the incident *horizontal switch* (denoted by ◯) to be ON.

### 1.2 Motivation

Unlike the traditional ASIC, the routing resources in an FPGA are prefabricated in the chip, and routing in an FPGA is performed

*The work of Yao-Wen Chang and Jai-Ming Lin was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-87-2215-E-009-041. E-mail: {ywchang, gis85504}@cis.nctu.edu.tw

†The work of D.F. Wong was partially supported by the Texas Advanced Research Program under Grant No. 003658288. E-mail: wong@cs.utexas.edu

by programming switches to make connections. The switches usually have high resistance and capacitance, and thus incur significant delays. To achieve better performance, each track should contain fewer horizontal switches (i.e., each segment has longer length and each track contains fewer segments). However, this would reduce routability and waste more wire lengths. On the other hand, if a track contains more horizontal switches (i.e., each segment has shorter length and each track contains more segments), nets can be routed with more flexibility and less waste of wire lengths. However, this would sacrifice performance. This trade-off between performance and routability presents a segmentation design problem: *How to determine a segmentation distribution to maximize the routability under performance constraints?*

**Example 1** *Figure 3 shows a set of three nets $n_1, n_2$, and $n_3$ to be routed in two different segmented routing channels with two tracks each. Each horizontal switch partitions a track into two segments. For example, in Figure 3(a), Track 1 consists of two segments [1, 2] and [3, 8], separated by the horizontal switch located between Columns 2 and 3. If each net can use at most one segment for routing, then nets $n_1, n_2$, and $n_3$ can not be routed simultaneously using the segmented channel shown in Figure 3(a); however, they can be routed if each net is allowed to use up to two segments. On the other hand, the three nets are always routable on the segmented channel shown in Figure 3(b). This example shows that segmentation designs could deeply influence the routability of an FPGA.*

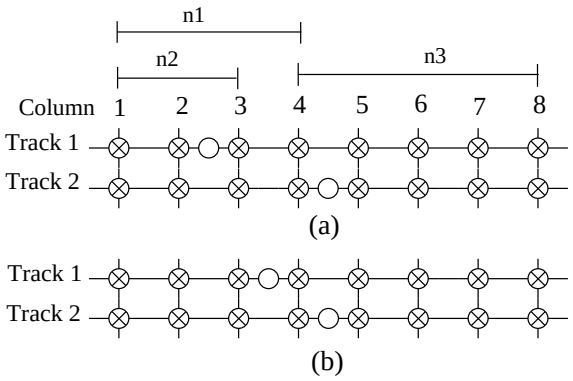


Figure 3: Routing three nets with two wire segments.

Rose and Hill in [10] emphasized that segmentation distribution would become a key challenge in large-scale FPGA design. They pointed out that even if one-million gate FPGAs became available today, physical design for these devices could be difficult because the routing delays and resource utilization could not be handled well and it is thus hard to realize the full potential of a large-scale FPGA. A well-designed segmentation can reduce not only routing delays but also waste of wire lengths. Therefore, the segmentation design problem will become even more important when the age of one-million gate is coming.

### 1.3 Previous Work

Channel segmentation designs have been studied to some degree in much of the literature [4, 7, 9, 11, 14]. El Gamal *et al.* showed that with appropriate arrangement of segment lengths, a segmented routing channel can achieve comparable routability to a customized routing channel [4]. For the channel segmentation design problem, Roy and Mehendale first presented a stochastic method which approximates a given segment length distribution [11]. Zhu and Wong in [14] also presented an algorithm for the channel segmentation design problem based on a stochastic analysis. Pedram *et al.* presented an analytical model for the design and analysis of effective segmented channel architectures [9].

Not much work has been reported for the two-dimensional segmentation design for the array-based FPGA. Further, existing work for the two-dimensional segmentation design is based on integration of the one-dimensional channel segmentation design. Zhu, Wong, and Chang observed that the two-dimensional segmentation design can be done in two stages: channel segmentation design followed by switch-module design [13]. Based on the similar idea, Mak and Wong recently employed a decomposition procedure and showed in details how the two-stage approach can be done [7].

### 1.4 Our Contributions

Most of the previously proposed methods are based on stochastic or analytical formulation. In this paper, we address a new direction for studying segmentation architectures. Our method is based on graph-theoretic formulation. We first formulate a net matching problem and present a polynomial-time optimal algorithm to solve the problem. Using the solution to the problem as a subroutine, we develop an effective and efficient matching-based algorithm for FPGA segmentation designs. Experimental results show that our method significantly outperforms the previous work. For example, our method achieves averages of 18.2% and 8.9% improvements in routability, compared with the work in [14] and [7], respectively. (Note that the most recent work [7] reports the best results among all previous work.) More importantly, our approaches are very flexible and can readily extend to higher-order segmentation designs (e.g., two- or three-dimensional segmentation design, etc) with only minor modifications. It should be pointed out that this scalability is crucial to the design of large-scale FPGAs.

## 2 Problem Formulation

In this paper, we will use the following notation:

- $L$: Length of a channel, measured in the number of columns. We number the columns from 1 to $L+1$.
- $T$: Number of tracks in the channel.
- $K$: Maximum number of segments allowed for routing a single net.
- $m$: Number of routing instances. A routing instance consists of a set of nets for routing.
- $n$: Number of nets in each routing instance.
- $d$: Number of dimensions for routing; $d = 1$ and 2 for channel and array-based routing/segmentation, respectively.

For the channel segmentation, each net is an *interval* which can be characterized by its leftmost and rightmost points. The leftmost and rightmost points of net $i$ are represented by $left_i$ and $right_i$, respectively. The *span* of net (interval) $i$ is from $left_i$ to $right_i$, denoted by $[left_i, right_i]$. One net *overlaps* another if the spans of the two nets intersect. A segment *covers* a net (interval) if the span of the net is within the bounds of the segment. A set $S$ of segments covers a routing instance $I$ (i.e., a set of nets) if for each net $i$ in $I$, there exists a segment $s$ in $S$ that covers $i$ and no two nets are covered by the same segment. For the array-based and higher-dimensional segmentation, the representation of a net and the definitions of span and cover need to be modified to consider the two-dimensional situations. We will discuss the modifications in Section 4.

For the *$K$-segment routing*, each net can use up to $K$ segments. For $K = 1$, a net can be routed on a segment as long as the segment covers the net. When one segment is assigned to a net, the segment is occupied and not allowed to be used for any other net. It is clear that if two nets overlap, they cannot be routed on the same track. For $K \geq 2$, each net can use multiple segments by programming corresponding horizontal switches to connect the segments. However, like 1-segment routing, each segment cannot be occupied by more than one net at a time.

*The $d$-dimensional segmentation design problem* is formulated as follows:

- **The $d$-Dimensional Segmentation Design Problem:** Given $L, T, K, m$ and $n$, design a $d$-dimensional segmentation to maximize the successful rate for $K$-segment routing.

For a fixed $K$, we refer to the problem as the *$d$-dimensional $K$-segmentation design problem*. When $K \geq 2$, it is also called the *$d$-dimensional multi-segmentation design problem*.

## 3 One-Dimensional Segmentation Design

We shall discuss our approach for one-dimensional segmentation design first. Our approach is motivated by the following observations. Given $m$ sets of routing instances, each with $n_i$ nets (intervals), $i = 1, \ldots, m$, designing a segmentation to maximize the successful rate for 1-segment routing is closely related to constructing a set $S$ of segments which can cover each of the $m$ sets of routing instances (one set at a time). It is not difficult to see that using such set $S$ of segments for 1-segment routing would result in 100% routing completion. However, there is usually a limitation on the number of tracks $T$ in a routing channel. Therefore, it is not always possible to construct a channel formed by all the segments in $S$. Nevertheless, the set $S$ of segments still gives a key insight into the optimal segmentation architecture for the given routing instances.

Since $S$ gives the optimal segmentation architecture, our goal is to construct a segmentation structure as close to $S$ as possible. Our method is based on graph-theoretic formulation. We first formulate a net matching problem to obtain a *most economical* set of segments that can cover each of two input routing instances. Based on the weighted bipartite matching theory, we present a polynomial-time optimal algorithm to solve the net matching problem. Using the solution to the problem as a subroutine, we then develop an effective bottom-up matching-based algorithm for the segmentation design for an arbitrary number of input routing instances. We shall first discuss the net matching problem.

### 3.1 The Net Matching Problem

Let $I$ be a finite set of (horizontal) intervals (nets). Let $i_1 = [left_1, right_1]$ and $i_2 = [left_2, right_2]$ be two overlapping intervals. We define $Merge(i_1, i_2)$ as the interval $i = [left, right]$, where $left = \min\{left_1, left_2\}$ and $right = \max\{right_1, right_2\}$. It is clear that the length of interval $i$, denoted by $len(i)$, is given by $right - left$ and the total length of all intervals in $I$, $Length(I)$, is given by $\sum_{i \in I} len(i)$.

Let $I$ and $J$ be two finite sets of intervals. A *net matching* $M$ between $I$ and $J$ is a set of ordered pairs of intersecting intervals $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$, where $A = \{i_1, i_2, \ldots, i_k\}$ and $B = \{j_1, j_2, \ldots, j_k\}$ are two sets of distinct intervals from $I$ and $J$, respectively. We can replace $i_1$ and $j_1$ by $Merge(i_1, j_1)$, replace $i_2$ and $j_2$ by $Merge(i_2, j_2)$, ..., and replace $i_k$ and $j_k$ by $Merge(i_k, j_k)$. After the replacement, the set of intervals $I \cup J$ becomes $Union(I, J) = (I - A) \cup (J - B) \cup \{Merge(i_1, j_1), Merge(i_2, j_2), \ldots, Merge(i_k, j_k)\}$.

*The Net Matching Problem* is described as follows:

- **The Net Matching Problem:** Given two finite sets $I$ and $J$ of intervals (nets), find a matching $M$ such that $Length(Union(I, J))$ is minimized.

Based on the weighted bipartite matching theory, we present a polynomial-time optimal algorithm for the Net Matching Problem. We reduce the problem to computing the maximum matching in a weighted bipartite graph. Given two finite sets $I$ and $J$ of intervals, we construct a weighted bipartite graph $G = (U, V, E)$ as follows. For each interval $i$ in $I$ ($j$ in $J$), we introduce a vertex $u_i$ ($v_j$) in the set $U$ ($V$) of vertices. For each pair of overlapping intervals, $p, q$, $p \in I$ and $q \in J$, connect $u_p$ to $v_q$ by an edge $e_{pq} = (u_p, v_q)$ with a weight computed by the weight function $\alpha : E \rightarrow Z^+$ defined as follows:

$$\alpha(e_{pq}) = \min\{right_p, right_q\} - \max\{left_p, left_q\}. \quad (1)$$

Then we can apply a maximum weighted bipartite matching algorithm [8] on $G$ to solve the Net Matching Problem optimally.

A *matching* $M$ of a *graph* $H = (V, E)$ is a subset of the edges with the property that no two edges of $M$ share the same vertex. Edges in $M$ are called *matched* edges; they are *unmatched*, otherwise. Let $Matched(I, J)$ be the set of the matched edges in a weighted bipartite matching on the graph induced by the finite sets $I$ and $J$ of intervals, and $Weight(F)$, $F \subseteq E$, be the total weight of the edges in $F$. We have the following lemma and theorem.

**Lemma 1** *$Length(Union(I, J)) = Length(I) + Length(J) - Weight(Matched(I, J))$.*

**Theorem 1** *The maximum bipartite weighted matching algorithm optimally solves the Net Matching Problem in $O((n_1 + n_2)^3)$ time, where $n_1$ and $n_2$ are the numbers of nets in the two input sets.*

**Example 2** *Figure 4(a) shows two sets $I = \{i_1, i_2, i_3, i_4\}$ and $J = \{j_1, j_2, j_3\}$ of intervals (nets). The induced weighted bipartite graph is given in Figure 4(b). The span of net $i$, $[left_i, right_i]$, is shown next to its corresponding vertex. The weight for each edge is computed by the function $\alpha$ and shown beside the edge. The maximum weighted bipartite matching $M$ between $U = \{u_1, u_2, u_3, u_4\}$ and $V = \{v_1, v_2, v_3\}$ is illustrated in Figure 4(b) by heavy lines. In this example, $M = \{(u_1, v_1), (u_2, v_3), (u_3, v_2)\}$. Note that $u_4$ is unmatched. Figure 4(c) shows the resulting configuration of replacing $i_1$ and $j_1$ by $Merge(i_1, j_1)$, $i_2$ and $j_3$ by $Merge(i_2, j_3)$, and $i_3$ and $j_2$ by $Merge(i_3, j_2)$. Let $l_1 = Merge(i_1, j_1), l_2 = Merge(i_2, j_3), l_3 = Merge(i_3, j_2)$, and $l_4 = i_4$. After the replacement, the set of intervals $I \cup J$ becomes $Union(I, J) = \{l_1, l_2, l_3, l_4\}$. The reader can verify that $Length(Union(I, J)) = len(l_1) + len(l_2) + len(l_3) + len(l_4) = 15$ is the minimum possible total union length for merging $I$ and $J$. (Note that $Length(Union(I, J)) = Length(I) + Length(J) - Weight(Matched(I, J)) = 15$).*

Note that $\alpha(e_{pq})$ gives the overlap length between intervals $p$ and $q$. Intuitively, this weight function measures the "similarity" between two intervals—the greater the weight, the more similar the two corresponding intervals. By merging intervals with greatest similarity, we can obtain a most economical (i.e., minimum total length) set of segments that covers each of two input interval sets.

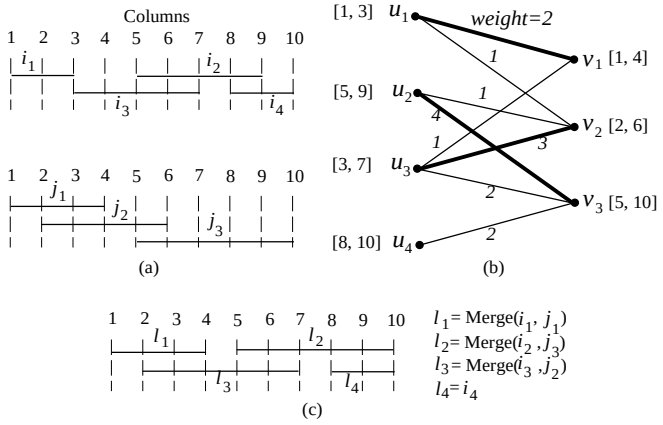


Figure 4: A matching-and-merging example. (a) Two sets of nets. (b) The corresponding weighted bipartite graph. (c) The matching result for the two sets of nets.

### 3.2 The Segmentation Design Algorithm

Our design algorithm consists of three stages: (1) the matching-and-merging stage, (2) the tuning stage, and (3) the filling stage. In the matching-and-merging stage, we repeatedly apply the aforementioned weighted bipartite matching algorithm to merge input routing instances and find a set $I$ of intervals that can cover each of the input routing instances. In the tuning stage, we find a set $I'$ of intervals from $I$, $I' \subseteq I$, which can be packed (routed) into $T$ tracks. In the filling stage, we determine the switch locations on the tracks and fill the empty space between each pair of intervals in the $T$ tracks to form a set of segments.

The matching-and-merging stage proceeds in a tree-like bottom-up manner. (The whole matching-and-merging process is illustrated in Figure 5.) Given $m$ routing instances $R_1, R_2, \ldots, R_m$, each with respective $n_1, n_2, \ldots, n_m$ nets, we apply the aforementioned weighted bipartite matching algorithm to merge $R_1$ and $R_2$, $R_3$ and $R_4$, $R_5$ and $R_6$, .... (See the procedure Match_and_Merge() in Lines 5 and 8 of Figure 6.) If $m$ is odd, then $R_m$ remains unmerged. After the merge, the number of resulting instances reduces

to $\lceil m/2 \rceil$. Then the same merging process repeats for the new $\lceil m/2 \rceil$ routing instances. The process proceeds level by level in a bottom-up manner until a final merged routing instance is obtained (see Figure 5).

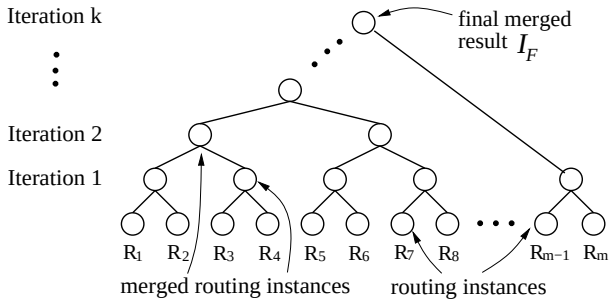


Figure 5: The matching process.

Let $I_F$ be the set of the intervals in the final merged routing instance. We have the following theorem.

**Theorem 2** *$I_F$ covers $R_i$, $\forall i, 1 \leq i \leq m$.*

By Theorem 2, using a set $S$ of segments covering $I_F$ for 1-segment routing can route all routing instances $R_1, R_2, \ldots, R_m$. As mentioned earlier, however, there is usually a limitation on the number of tracks $T$ in a routing channel. Therefore, it is not always possible to construct a channel formed by all the segments in $S$.

In the tuning and the filling stages, we construct a segmentation of $T$ tracks from the final merged routing instance $I_F$. First, we apply the basic left-edge algorithm [6] to route the intervals in $I_F$. (See the procedure Route_by_Left_Edge() in Line 11 of Figure 6.) We then sort the resulting tracks in the non-increasing order of their total lengths occupied by the intervals. The first $T$ tracks are chosen for further construction, and the tuning stage is done. (See the procedure Tune_Tracks() in Line 12 of Figure 6.) In the filling stage, we determine the switch locations on the tracks and fill the empty space between each pair of intervals in the $T$ tracks to form a set of segments. To optimize the routability of a designed segmentation, it is important to consider the positions for placing horizontal switches. We have the following theorem to guide the placement of horizontal switches.

**Theorem 3** *For the uniform net distribution, a segmented routing track can cover the maximum number of nets when switches are evenly spaced on the track.*

Therefore, by Theorem 3, if there is an empty space between two intervals on a track, we shall place a horizontal switch on the position that makes the two resulting *segments* most balanced in length. The procedure Fill_Space() listed in Line 13 of Figure 6 is based on Theorem 3 to find optimal positions for placing horizontal switches. The whole segmentation design algorithm is summarized in Figure 6.

**Theorem 4** *Algorithm Seg_Designer runs in $O(m^3 n^3)$ time, where $m$ is the number of input routing instances and $n$ is the maximum number of nets in a routing instance.*

For $K$-segmentation design ($K \geq 2$), all we need to do is splitting each segment into $K$ sections of equal length right after the above-mentioned procedures. However, since the minimum length of a segment is one, it is impossible to partition an interval of length smaller than $2K-1$ into $K$ segments. Specifically, we can partition an interval of length $l$ into at most $\lceil l/2 \rceil$ segments.

**Algorithm:** Seg_Designer($m$, $R[i]$, $T$)
**Input:** $m$—Number of routing instances;
$R[m]$—$R[i]$ is the $i$-th routing instance, $0 \leq i \leq m-1$;
$T$—Maximum number of tracks in the channel.
**Output:** $S$—The designed segmentation.

```
Stage 1: Matching and Merging
1   iteration ← ⌈log2 m⌉;
2   for i ← 1 to iteration do
3       if (m is even)
4           for j ← 0 to m/2 − 1 do
5               R[j] ← Match_and_Merge(R[2j], R[2j + 1]);
6       else
7           for j ← 0 to ⌊m/2⌋ − 1 do
8               R[j] ← Match_and_Merge(R[2j], R[2j + 1]);
9           R[j + 1] ← R[m − 1];
10      m ← ⌈m/2⌉;
Stage 2: Tuning
11  Track[t] ← Route_by_Left_Edge(R[0]);
12  Track[T] ← Tune_Tracks(Track[t]);
Stage 3: Filling
13  S ← Fill_Space(Track[T]);
14  return S.
```

Figure 6: The algorithm for segmentation design.

# 4 Two-Dimensional Segmentation Design

Our approaches are very flexible and can readily extend to higher-order segmentation designs (e.g., two- or three-dimensional segmentation design, etc) with only minor modifications.

We briefly describe how to extend the matching-based algorithm to the two-dimensional segemtation design. (For the higher-order segmentation design, the extension is similar.) The most significant difference between designs for one- and two-dimensional segmentations lies in the representation of a net. In the one-dimensional segmentation design, we can simply represent a net $x$ by its span $[left_x, right_x]$. The representation of a net in two-dimensional channels is more sophisticated. (See Figure 7 for the representation.) For a $p \times q$ (number of logic modules) array-based FPGA, there are $p-1$ horizontal and $q-1$ vertical routing channels. We label the channels in an array-based FPGA $1, 2, \ldots, p-1$ from the top to the bottom, and $p, p+1, \ldots, p+q-2$ from the left to the right. If a net runs through more than one channel (and/or more than one track in a channel), we divide the net into a set of subnets, one subnet for each channel (and/or for each track). Each subnet $x_i$ in channel $c_{x_i}$ is represented by the three tuple $[left_{x_i}, right_{x_i}, c_{x_i}]$, where $left_{x_i}$ and $right_{x_i}$ are the leftmost and rightmost points of the subnet. Then, a net can be represented by a *set* of the three tuples for its subnets.

The Net Matching Problem described in Section 3 is extended to handle two-dimensional nets and called *The Two-Dimensional Net Matching Problem*. The Two-Dimensional Net Matching Problem can also be optimally solved by reducing the problem to computing the maximum matching in a weighted bipartite graph. Given two finite sets $I$ and $J$ of nets, we construct a weighted bipartite graph $G = (U, V, E)$ as follows. For each net $i$ in $I$ ($j$ in $J$), we introduce a vertex $u_i$ ($v_j$) in the set $U$ ($V$) of vertices. One net *overlaps* another if the spans of any pair of subnets of the two nets intersect. For each pair of overlapping nets, $p, q$, $p \in I$ and $q \in J$, with the respective subnets $\{[left_{p_1}, right_{p_1}, c_{p_1}], \ldots, [left_{p_h}, right_{p_h}, c_{p_h}]\}$ and $\{[left_{q_1}, right_{q_1}, c_{q_1}], \ldots, [left_{q_k}, right_{q_k}, c_{q_k}]\}$, connect $u_p$ to $v_q$ by an edge $e_{pq} = (u_p, v_q)$ with a weight computed by the weight

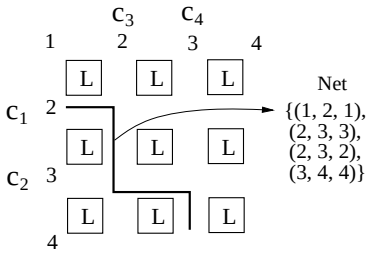


Figure 7: Represetation of a net in an array-based FPGA.

function $\beta : E \rightarrow Z^+$ defined as follows:

$$\beta(e_{pq}) = \sum_{c_{p_i}=c_{q_j}, p_i, q_j} \Phi(p_i, q_j), \quad (2)$$

where $\Phi(p_i, q_j) = \min\{right_{p_i}, right_{q_j}\} - \max\{left_{p_i}, left_{q_j}\}$. Then we can apply a maximum weighted bipartite matching algorithm [8] on $G$ to solve the net matching problem optimally. We have the following theorem.

**Theorem 5** *The maximum bipartite weighted matching algorithm optimally solves the Two-Dimensional Net Matching Problem in $O((n_1 + n_2)^3 + r_1 r_2)$ time, where $n_1$ and $n_2$ ($r_1$ and $r_2$) are the numbers of nets (subnets) in the two input sets.*

The algorithm for the two-dimensional segmentation design, 2D_Seg_Designer, is very similar to that for the one-dimensional design described in the previous section. The design algorithm for two-dimensional segmentation also consists of the three stages: (1) the matching-and-merging stage, (2) the tuning stage, and (3) the filling stage. In the matching-and-merging stage, we repeatedly apply the aforementioned weighted bipartite matching algorithm to merge two-dimensional input routing instances to find a set $I$ of nets that can cover each of the input routing instances. In the tuning and the filling stages, we construct tracks channel by channel by considering the *subnets* of $I$ in each channel obtained in the matching-and-merging stage. For the filling stage, the process is similar to that for the one-dimensional segmentation design. The following theorem gives the time complexity of our algorithm.

**Theorem 6** *Algorithm 2D_Seg_Designer runs in $O(m^3(n^3 + r^2))$ time, where $m$ is the number of input routing instances and $n(r)$ is the maximum number of nets (subnets) in a routing instance.*

## 5 Experimental Results

We implemented our segmentation design algorithms in the C++ programming language on a PC with a Pentium 166 microprocessor and 32 MB RAM. The weighted bipartite matching code was adopted from the public LEDA package. The routability of the architectures designed by our algorithms was tested using the 1- and the multi-segment routing algorithms by Zhu and Wong [14]. In addition to the notation mentioned in Section 2, the following notation is also needed to explain our experimental procedures.

- $D$: Maximum number of net terminals at a column.
- $f(l)$: Probability that a net is of length $l$.

The input routing instances were generated by the programs used in [7] and [14]. The first set of ten net distributions is based on the parameters $L = 100$, $T = 36$, and $D = 12$ which are close to the row-based architectures used by Actel FPGAs [1]. Distributions $D_i$, $i = 1, 2, \ldots, 7$ are defined as follows. If $f(l) = (p_1, p_2, p_3, p_4, p_5)$, then the probability that a net has length between $0.2(j-1)L$ and $0.2jL$ is equal to $p_j / \sum_{1 \le k \le 5} p_k$. "Ge", "No", and "Po" are geometric, normal, and Poisson distributions, respectively. For each net distribution, 300 routing instances were generated.

The ratio of routing success was measured by the *threshold density* $d_T$ defined in [14]. $d_T$ means the smallest channel density $d$ such that less than 90% of the channels with density $d$ in the distribution are successfully routed.

Table 1 lists the respective comparisons for one- and two-segment routing between our designs and those in Zhu and Wong [14] based on the parameters $L = 100, T = 36$, and $D = 12$ which were used in [14]. The results show that our designs outperform those in [14] by *averages* of 24.0% and 12.9% improvements in routability for one- and two-segment routing, respectively.

The channel segmentation designs in [7] were targeted for array-based FPGAs. The parameters used for net distributions were $L = 20, T = 18$, and $D = 6$ for two-segment design and $L = 50, T = 24$, and $D = 8$ for three-segment design. The results, reported in Table 2, show that our method significantly outperforms the previous work in [7] and [14]. Our designs achieve *averages* of 17.9% and 8.9% improvements in routability, compared with the work in [14] and the most recent work in [7], respectively. Figure 8 shows our 1-segment channel segmentation design for distribution $D_1$ using the parameters $L = 100, T = 36$, $D = 12$, and $K = 1$.

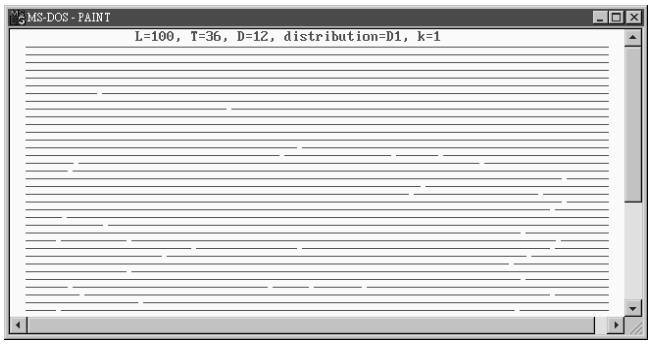


Figure 8: A channel segmentation designed by our algorithm ($L = 100, T = 36, D = 12, K = 1$, Distribution $D_1$).

Our design algorithms are quite efficient. The empirical run times for the largest set of designs ($L = 100, T = 36, D = 12$, and $K = 2$) ranged from 10 sec for Distribution $D_2$ to 78 sec for Distribution $D_7$ (with an average run time of about 30 sec). Although the theoretic analysis gives $O(m^3 n^3)$-time complexity for our algorithm, the empirical run time for the $n$ term is close to $O(n \lg n)$, instead of $O(n^3)$. The reason is that most of the nets in two input routing instances were merged together. Therefore, the number of nets in a merged instance grew only linearly, instead of exponentially. In Figure 9, the average number of nets per routing instance is plotted as a function of the number of iterations (in the matching-and-merging stage) for each of the ten distributions listed in Table 1. The curves in Figure 9 exhibit the linearity of the empirical growth rates for the average number of nets per routing instance.

## 6 Conclusion

We have presented a new direction based on the graph-matching formulation for studying FPGA segmentation design. The approach is so effective, efficient, and flexible that it can easily extend to more sophisticated segmentation designs. Future work lies in the extension of higher-order segmentation designs. Also, the results of the matching-and-merging stage could be sensitive to the pairing of input routing instances. The general graph-matching algorithm might be a promising solution to the best pairing of the routing instances.

## Acknowledgments

The authors would like to thank Kai Zhu and Wai Kei Mak for providing us with their packages for segmentation designs.

## References

[1] Actel Corporation, *FPGA Data Book and Design Guide*, 1996.

| | $f(l)$ | One-segment routing ($K = 1$) | | | | Two-segment routing ($K = 2$) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | [13] | | Ours | | [13] | | Ours | |
| | | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ |
| $D_1$ | (1, 1, 1, 1, 1) | 24 | 0.67 | 31 | 0.86 | 29 | 0.81 | 33 | 0.92 |
| $D_2$ | (.1, .3, .5, .8, 1) | 32 | 0.89 | 34 | 0.94 | 34 | 0.94 | 35 | 0.97 |
| $D_3$ | (1, .8, .5, .3, .1) | 23 | 0.64 | 28 | 0.78 | 28 | 0.78 | 32 | 0.89 |
| $D_4$ | (1, .5, .3, .1, 0) | 26 | 0.72 | 27 | 0.75 | 26 | 0.72 | 32 | 0.89 |
| $D_5$ | (1, .5, .3, .5, 1) | 20 | 0.56 | 30 | 0.83 | 27 | 0.75 | 33 | 0.92 |
| $D_6$ | (.2, .5, 1, .5, .2) | 29 | 0.81 | 33 | 0.92 | 33 | 0.92 | 35 | 0.97 |
| $D_7$ | (1, .2, .1, 0, 0) | 22 | 0.61 | 27 | 0.75 | 22 | 0.61 | 28 | 0.78 |
| Ge | $\gamma^l$,$\gamma = 0.95$ | 21 | 0.58 | 28 | 0.78 | 25 | 0.70 | 30 | 0.83 |
| No | $\mu = 35$, $\sigma^2 = 100$ | 25 | 0.70 | 31 | 0.86 | 32 | 0.89 | 33 | 0.92 |
| Po | $\lambda^l e^{-\lambda}/l!$, $\lambda = 20.0$ | 20 | 0.56 | 31 | 0.86 | 30 | 0.83 | 32 | 0.89 |
| Avg | | 24.2 | 0.674 | 30.0 | 0.833 | 28.6 | 0.795 | 32.3 | 0.889 |

Table 1: One- and two-segment routing results ($L = 100, T = 36, D = 12$).

| | $f(l)$ | Two-segment routing ($K = 2$) | | | | | | Three-segment routing ($K = 3$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | [14] | | [7] | | Ours | | [14] | | [7] | | Ours | |
| | | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ | $d_T$ | $d_T/T$ |
| $b_1$ | (1, 1, 1, 1, 1) | 15 | 0.83 | 17 | 0.94 | 17 | 0.94 | 21 | 0.88 | 22 | 0.92 | 23 | 0.96 |
| $b_2$ | (1, .8, .5, .3, .1) | 14 | 0.78 | 15 | 0.83 | 16 | 0.89 | 17 | 0.71 | 21 | 0.88 | 24 | 1.00 |
| $b_3$ | (1, .5, .3, .1, 0) | 13 | 0.72 | 14 | 0.78 | 17 | 0.94 | 18 | 0.75 | 21 | 0.88 | 21 | 0.88 |
| $b_4$ | (1, .5, .3, .5, 1) | 13 | 0.72 | 16 | 0.89 | 16 | 0.89 | 19 | 0.79 | 22 | 0.92 | 21 | 0.88 |
| $b_5$ | (.2, .5, 1, .5, .2) | 16 | 0.89 | 15 | 0.83 | 17 | 0.94 | 22 | 0.92 | 22 | 0.92 | 22 | 0.92 |
| $b_6$ | (1, .2, .1, 0, 0) | 11 | 0.61 | 14 | 0.78 | 17 | 0.94 | 17 | 0.71 | 20 | 0.83 | 22 | 0.92 |
| Ge | $\gamma^l$,$\gamma = 0.7$ | 12 | 0.67 | 13 | 0.72 | 16 | 0.89 | 20 | 0.83 | 20 | 0.83 | 22 | 0.92 |
| No | $\mu = 4$, $\sigma^2 = 10$ | 16 | 0.89 | 15 | 0.83 | 17 | 0.94 | 21 | 0.88 | 20 | 0.83 | 21 | 0.88 |
| Po | $\lambda^l e^{-\lambda}/l!$, $\lambda = 3.0$ | 13 | 0.72 | 15 | 0.83 | 16 | 0.89 | 19 | 0.79 | 19 | 0.79 | 23 | 0.96 |
| Avg | | 13.7 | 0.759 | 14.9 | 0.826 | 16.6 | 0.917 | 19.3 | 0.807 | 20.8 | 0.867 | 22.1 | 0.924 |

Table 2: Two- and three- segment routing results (for $K = 2$: $L = 20, T = 18, D = 6$; for $K = 3$: $L = 50, T = 24, D = 8$).

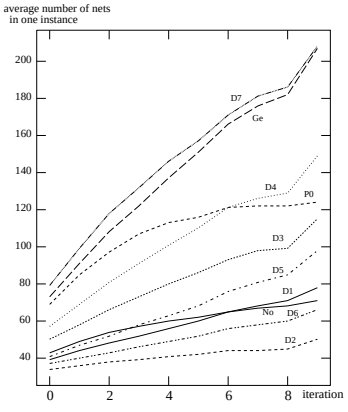


Figure 9: Average number of nets in one instance at each iteration.


[2] Lucent Technologies, *ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series Field-Programmable Gate Arrays*, August 1996.

[3] A. El Gamal *et al.*, "An Architecture for Electrically Configurable Gate Array," *IEEE J. of Solid State Circuits*, Vol. 24, No 2, April 1989, pp 394-398.

[4] A. El Gamal, J. Greene, and V. Roychowdhury, "Segmented Channel Routing is Nearly as Efficient as Channel Routing (and Just as Hard)," *Advanced Research in VLSI*, pp. 193–221, 1991.

[5] M. Khellah, S. Brown, and Z. Vranesic, "Minimizing interconnection delays in array-based FPGAs," in *Proc. of IEEE Custom Integrated Circ. Conf.*, pp. 181–184, May 1994.

[6] M. J. Lorenzetti and D. S. Baeder, Chapter 5: *Routing* in *Physical Design Automation of VLSI Systems*, Edited by B. Preas and M. Lorenzetti, Benjamin/Cummings, 1988.

[7] W. K. Mak and D. F. Wong, "Channel Segmentation Design for Symmetrical FPGAs," *Proc. of IEEE ICCD*, Oct. 1997.

[8] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., 1982,

[9] M. Pedram, B. S. Nobandegani, and B.T. Preas, "Design and analysis of segmented routing channels for row-based FPGAs," *IEEE Trans. on CAD*, Vol. 13, No. 12, pp. 1470–1479, Dec. 1994,

[10] J. Rose and D. Hill, "Architectural and Physical Design Challenges for One-Million Gate FPGAs and Beyond," *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, pp. 129–132, Feb. 1997.

[11] K. Roy and M. Mehendale, "Optimization of Channel Segmentation for Channelled Architecture FPGAs," in *Proc. of IEEE Custom Integrated Circ. Conf.*, pp. 4.4.1–4.4.4, May 1992.

[12] Xilinx Inc., *The Programmable Logic Data Book*, 1996.

[13] K. Zhu, D. F. Wong, and Y.-W. Chang, "Switch module design with application to two-dimensional segmentation design," *Proc. of IEEE/ACM ICCAD*, pp. 480–485, 1993.

[14] K. Zhu and D. F. Wong, "Segmented Channel Segmentation Design for Row-Based FPGAs," *Proc. of IEEE/ACM ICCAD*, pp. 26–29, 1992.