PADE: A High-Performance Placer with Automatic Datapath Extraction and Evaluation through High-Dimensional Data Learning

Samuel Ward, Duo Ding, David Z. Pan,
ECE Dept. The University of Texas at Austin, Austin, TX 78712
{wardsi}@utexas.edu, {ding,dpan}@cerc.utexas.edu

ABSTRACT

This work presents PADE, a new placer with automatic datapath extraction and evaluation. PADE applies novel data learning techniques to train, predict, and evaluate potential datapaths using high-dimensional data such as netlist symmetrical structures, initial placement hints and relative area. Extracted datapaths are mapped to bit-stack structures that are aligned and simultaneously placed with the random logic. Results show at least 7% average total Half-Perimeter Wire Length (HPWL) and 12% Steiner Wire Length (StWL) improvements on industrial hybrid benchmarks and at least 2% average total HPWL and 3% StWL improvements on ISPD 2005 contest benchmarks. To the best of our knowledge, this is the first attempt to link data learning, datapath extraction with evaluation, and placement and has the tremendous potential for pushing placement state-of-the-art for modern circuits which have datapath and random logics.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuits**]: Design Aids—*Placement and Routing*

General Terms

Design

Keywords

Datapath, Placement, Extraction, Physical Design

1. INTRODUCTION

Advancements in random logic placement have been impressive over the last few years with modern placers able to handle over one million placeable objects in minutes (e.g., [1]). Typically, these placers optimize the half-perimeter wire length (HPWL) objective standardized by the 2005 ISPD Placement contests [2] and is a good indicator of placement quality for random logic designs [3]. Unlike random logic, datapath logic generally is characterized by a high degree of bit-wise parallelism [4] (often called bit-stack) that modern placers have shown to be suboptimal [5]. This is partly due to the inaccuracy of the HPWL model when compared to the Steiner wire length (StWL) [6].

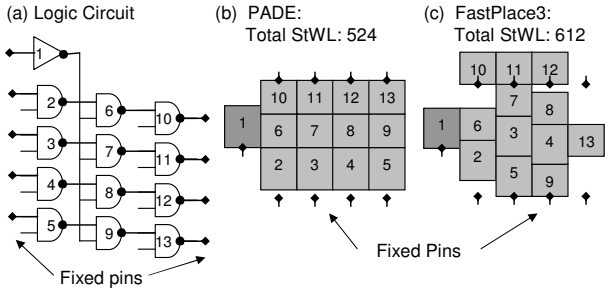

Figure 1: PADE placement example showing a 14% StWL improvement compared to FastPlace3 [7].

Figure 1 shows a toy example where modern placers are not able to handle datapaths effectively. Figure 1(a) displays the datapath circuit, where the input and output pins are fixed. Cell 1 is an inverter driving four $NAND2$ gates and there are three bit-stacks corresponding to cells: $(\{2, 3, 4, 5\}, \{6, 7, 8, 9\}$, and $\{10, 11, 12, 13\}$. For clarity, Fig. 1(b-c) display fixed pin locations. Fig. 1(b) displays the PADE placement solution solution. In this case, each bit-stack is tightly packed and aligned producing an StWL solution of 524. Figure 1(c) displays the placement solution from FastPlace3 [7] where the bit-stack is not carefully aligned producing StWL of 612. In fact, with only thirteen cells, the StWL solution in 1(c) is over 14% worse than that in 1(b). In designs where there are many embedded datapaths, extracting the datapath and placing them with random logics properly has the potential for significant improvement in the overall StWL. Datapath extraction techniques in the past generally focused on functional or structural levels. Functional regularity extraction identifies logically equivalent subcircuits within a netlist that are then handled separately during placement. One example of this method is developed in [8] where a large set of templates are generated and used to search for datapath logic before placement. Another example is the hash-based approach of [9]. Structural datapath extraction techniques have focused on developing a regularity metric to represent the datapath. In [4], the datapath extraction consists of a decomposition of the netlist into a set of stages and a set of slices with one cell occurring in exactly one stage set and one slice set. The extraction algorithm expands in search-waves through the network using the regularity metric to determine the expansion direction. More recently, [10] developed a method for extracting structure within a design with the assumption that the placement distance between a pair of cells is related to the graph distance between them. Nets are weighted in a shortest path computation by assuming the distance between two cells is related to the degree of the net connecting them. Then, by extracting "corner" cells and fixing them in place, the maximum

[0]This work is supported in part by IBM Faculty Award and Oracle

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

distance of the other cells can be calculated.

These previous datapath extraction algorithms, which only use functional or structure information, are not effective for modern large-scale hybrid datapath/random logic circuits with many pre-placed IP blocks: (1) the placement blockage could force the bit-stacks to be placed away from adjacent logic causing wirelegnth degradation; (2) it may give out too many or the wrong bit-stacks which would also adversely affect the overall placement quality. Compounding the problem, as shown in [11], a dedicated datapath placer often overly constrains the random logic placer. These problems lead to the general industry practice of manually designing the datapath because of the possible significant timing and wire-length improvement by careful alignment and packing of the bit-stack. However, increasing design sizes and shortening turn-around-time demand a consolidated automated datapath extraction and placement framework.

In this paper, we propose PADE, a new placer with automatic datapath extraction which can handle large scale designs mixed with random and datapath circuits. PADE will evaluate and rank all the first-order, important data paths, and optimize them along with general-purpose wirelength driven placement[1]. The key contributions of this paper include:

- We develop a novel high-dimensional data learning, extraction, and evaluation algorithm for datapath extraction in PADE. It considers not only logic structures, but also placement hints from initial global placement results.
- We develop an optimal algorithm for bit-stack selection (to be used for guiding data-path aware placement) using integer linear programming.
- PADE has demonstrated significantly better results than previous state-of-the-art placers on both hybrid industrial designs which contain both random logics and datapaths, and even the ISPD 2005 placement benchmarks where structured datapath logics were not intended.

Section 2 outlines the overall flow and section 3 details the high-dimensional extraction data. Section 4 describes the model training and cluster evaluation and section 5 describes the binary integer programming bit-stack assignment technique. Experimental results are presented in section 6 followed by conclusions in section 7.

2. OVERALL FLOW OF PADE PLACER

Given a netlist $N = (V, E)$ with nodes V and nets E, placement obtains locations (x_i, y_i) for all movable nodes, such that the area of nodes within the placement boundary does not exceed the area of cell sites in that region. With $\vec{x}, \vec{y} = \{x_i, y_i\}$, HPWL is defined as: $HPWL(\vec{x}, \vec{y}) = HPWL(\vec{x}) + HPWL(\vec{y})$ where $HPWL(\vec{x}) = \sum_{e \subset E}[MAXx_i - MINx_i]$. Modern placers often approximate HPWL by a differentiable function using the quadratic objective, defined as:

$$\Phi_G(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j}[(x_i - x_j)^2 + (y_i - y_j)^2] \qquad (1)$$

From Equation 1, (x_i, y_i) represents the coordinates of cell i, and $w_{i,j}$ represents the weight between cells i and j. A datapath netlist with p bit-stacks, each bit-stack B_k $0 < k < P$ is a disjoint set of cells $B_k \subset V$, describing the bit-wise parallelism present in the netlist. Representing the datapath as a set of cells in this manner enables implicit StWL optimization through forced alignment as presented in [6].

[1]The name for PADE placer is also inspired by the famous Pade approximation which is widely used in model order reduction, as they share the same principle to extract the first-order effects.

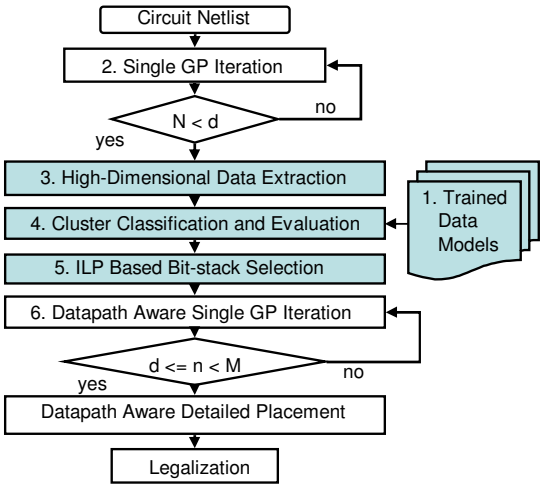

Figure 2: Overall flow of the PADE placer.

PADE is a modern force-directed global placer extending the SAPT [6] placer where $w_{i,j}$ is given by the Bound2Bound net model [12] and it utilizes a detailed placer similar to FastPlace3 [7].

Briefly, SAPT is a datapath aware placer requiring manual definition of both the bitstack and the datapath direction. For each bit-stack B_k, an alignment net is inserted, similar to a pseudo-net but remains persistent between placement iterations, connecting each cell in B_k. The alignment net is manipulated through the use of skewed weighting on $w_{i,j}$ and modified fixed-point insertion making it possible to introduce an alignment constraint to a predefined group of cells within the linear solver during global placement. A gradually increasing application of this constraint aligns each bit-stack through consecutive global placement iterations, which enables a unified placement framework that simultaneously places datapath and random logic cells without over constraining the placer. Additionally, during detailed placement the placer maintains that alignment by constraining cell movements along the bit-stacks.

The overall flow of the PADE placer is shown in Fig. 2 with the novel datapath training, extraction, evaluation and datapath bit-stack selection stages shaded. To properly handle the extraction of datapath structures, a compact and high performance knowledge base is proposed to identify datapath patterns from non-datapath logics and to evaluate the placement quality of these patterns. The knowledge base, step 1. in Fig. 2, is constructed via performing advanced data learning algorithms over a set of baseline design benchmarks after placement. By construction, it explores the placement database and captures the special characteristics that strongly correlate to datapath patterns, such as netlist connectivity automorphisms. Once these characteristics are captured and extracted, a complex decision diagram is built at a one time cost, which can later be applied to classify datapath netlists and non-datapath logics very rapidly. This knowledge base allows special treatment of the datapath logics in the placement stage without degrading the performance of the rest of the design. Additionally, these models are generic and can be applied to any circuit.

Step 2., a single global placement (GP) iteration, is one standard iteration of a force-directed placer that includes pseudo net insertion, linear system solver and fixed point generation. Let M, $0 \leq n \leq M$, be the upper bound on the global placement iterations and d an intermediate point at which time a prediction on the datapath will be made. Integrating the datapath extraction during global placement (GP) instead of before allows for enhanced prediction accuracy by taking into account physical characteristics in

addition to netlist regularity measures.

Step 3 extracts the high-dimensional features from the netlist and step 4 classifies and then evaluates the datapath candidates. For all identified datapath logics, step 5 uses an ILP formulation to map cells to a bit-stack and then inserts the alignment nets. Step 6 performs datapath aware global placement with bit-stacks aligned during the following global placement iterations. The flow completes with datapath aware detailed placement and legalization as described in [6].

3. HIGH-DIMENSIONAL EXTRACTION

In this work, both graph-based and physical features are analyzed and extracted from the netlist mapping a set of parameters most critical and sensitive to datapath logics. Effective features create differentiation between random and datapath logic allowing the patterns extracted on the training set to effectively classify datapath structures in new circuits and predict the direction of the datapath. The first step in this process is to generate candidate clusters of the original netlist in which to search for datapath structures.

3.1 Seed-Based Connectivity Clustering

The connectivity based clustering stage prepares the data to analyze and extract datapath structures from. The goal is to find clusters exhibiting the structure we are looking for. Extending the seed growth method proposed in [13], the clustering method creates k clusters. It maximizes the ratio of the external to internal force of a cluster C_i, where C_i $0 \leq i < k$ indicates a group of vertices, while maintaining a maximum logic depth threshold. The external force is defined as the summation of the edge weights of nets with at least one vertex outside and one inside C_i and the internal force is defined as the summation of all internal cluster weight connections. The weight $w(u, j)$ is determined by the net model used, in this case a clique representation, where a connection c for a given edge e, $w(c) = w(e)/((|e|-1)|e|)$. The connectivity between neighbor node u and cluster C_i is given by $conn(u, C_i) = \sum_{j \subset C_i} w(u, j)$ where suitable seed nodes are those with a large net degree.

In each subsequent pass, the neighbor node with the largest connectivity $conn$ is added to the cluster C_i while keeping the internal force of the cluster as large as possible. Once a cluster's node cardinality reaches the threshold value or the entire netlist is clustered, the high-dimensional features described in the next subsection are extracted from each cluster C_i and then the cluster is classified as a datapath or random logic cluster.

3.2 Automorphism Feature Extraction

In this section we describe the graph features used to differenciate datapath and random logic. One of the fundamental observations in this work is that datapath logic contains a high degree of graph automorphism. An automorphism of a graph, a form of symmetry, preserves the edge - vertex connectivity of the graph while mapping onto itself [2]. That is, an automorphism is a graph isomorphism from G to itself.

DEFINITION 1. *Automorphism: An automorphism of a graph $G = (V, E)$ is a permutation σ of the vertex set V, such that the pair of vertices (u, v) form an edge if and only if the pair $(\sigma(u), \sigma(v))$ also form an edge.*

Automorphism Group: The set of automorphisms of a given graph forms the automorphism group of the graph and is denoted by $Aut(G)$. The set $S \subseteq Aut(G)$ of generators for $Aut(G)$ is a set whereby combining elements of S generates every non-identity permutation in $Aut(G)$.

[2]Assuming reader familiarity with graph automorphisms and permutations. Please refer to [14] for further details.

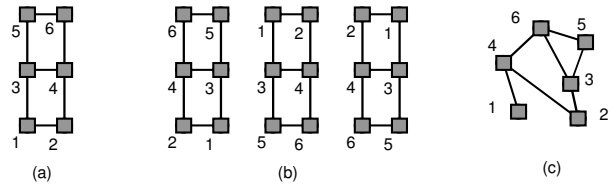

Figure 3: Graph automorphism example showing the original graph in (a) with each of the automorphisms in (b). A random netlist is shown in (c) with only trivial automorphisms.

DEFINITION 2. *Generator Set: A generator set of a group is a subset such that every element of the group can be expressed as the combination, under the group operation, of finitely many elements of the subset and their inverses.*

As an example, Fig. 3(a) displays a graph G with six labeled nodes and seven edges. The *automorphism feature* is represented with a seventeen parameter vector $(|Aut(G)|, \Delta(2:18),)$ where $|Aut(G)|$ is the cardinality of the automorphism group S_i for cluster C_i. The last sixteen parameters, $\Delta(2:18)$, are from the frequency table of the size of each automorphism.

As an example in Fig. 3, the graph G has a total of four automorphisms and two generators ($|S|$ = 2, with $|Aut(G)| = 4$). The first automorphism, $G(1, 2, 3, 4, 5, 6)$, corresponds to itself and three additional automorphisms $G(2, 1, 4, 3, 6, 5)$ G flipped left-right, $G(5, 6, 3, 4, 1, 2)$ G flipped up-down, and $G(6, 5, 4, 3, 2, 1)$ G flipped left-right and up-down, displayed in Fig. 3(b). The nontrivial generator set S of G is $(1, 5)(2, 6)$ and $(1, 2)(3, 4)(5, 6)$. As this example shows, the symmetry of the graph along with the generator group provides possible bit-stack candidates including: $(1, 2)$, $(5, 6)$ or $(1, 3, 5)$, $(2, 4, 6)$.

Figure 3(c) displays a random logic netlist also with six nodes and seven edges. Unlike the clear symmetry present in Fig. 3(a), Fig. 3(c) contains no non-trivial automorphisms. In fact, this is a fundamental observation holding true for random logic netlists in general. Thus, the automorphism generators of structured logic appear very differently than the automorphism generators of random logic netlists enabling sufficient differentiation as a datapath feature.

3.3 Physical Aware Feature Extraction using Placement Hints

Graph automorphism features alone do not capture the physical nature of the placement problem, a fundamental shortcomming of prior extraction techniques. Global placement has merit in wirelength optimization, which shall be used for improved classification. Thus physical features extracted after the first few passes define the following attributes.

Let a_i^c be the sum of the total cell area within cluster C_i, w_i^c be the bounding box width from the placement for C_i, h_i^c be the bounding box height and finally r_i^c be the ratio $a_i^c/(w_i^c + h_i^c)$. This physical information helps to characterize the amount of spreading and the initial cell locations for each C_i. Dense clusters indicate tightly packed logic and possibly the need for improved placement whereas sparse logic is generally less likely to improve from being passed to the datapath placer. In the next section, the training steps for building the model and the process to evaluate each cluster is described.

4. DATAPATH MODEL TRAINING AND CLUSTER EVALUATION

To classify and evaluate the datapath patterns in each cluster, we propose to combine data learning algorithms Support Vector Machine (SVM) and Neural Network (NN) to build compact and

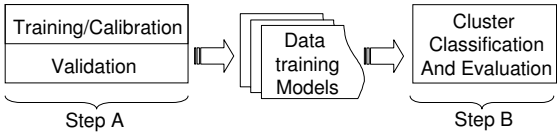

Figure 4: Major steps to build and apply the learning models

run-time efficient models as shown in Fig. 4. SVM calculates a hyperplane boundary with maximum separation margin in-between of datapath and non-datapath. Only the critical information on the separation boundaries is preserved (the support vectors SV). All SV's are involved in the decision (score) calculation. For better quality, we combine a soft-error tolerant SVM and a special working set selection method [15]. NN works through configuring complex networks of neurons to achieve a high dimensional decision diagram-like data structure given training samples and decision hints. We employ a resilient backward propagation method based on iterative sub-gradient updates. To quantify the learning performance, we define the following two types of accuracies:

DEFINITION 3. *Datapath evaluation accuracy: the rate of correctly detected datapath (datapath-like) patterns over the total number of actual datapath structures.*

DEFINITION 4. *Non-datapath evaluation accuracy: the rate of correctly detected non-datapath (e.g., random logic) patterns over the total number of non-datapath structures processed.*

The optimization objective for both SVM and NN is to maximize the evaluation accuracies of datapath and non-datapath patterns, or equivalently, to minimize the mean square errors for both classes of pattern evaluation. This is achieved in two steps: Step A and B as shown in Fig. 4.

4.1 Training, Calibration and Validation

In Step A, we first apply data learning algorithms over a relatively small set of design patterns with known datapath information under the guidance of placement as hints. Since they are built a priori at a one time cost, the CPU run-time penalty is negligible. There are 3 major procedures involved in this step: (1) training is the process where the learning algorithms optimize both datapath and non-datapath accuracies; (2) calibration process further improves the accuracies, e.g., via properly selecting the separation threshold in (1); (3) validation process is performed over a relatively large set of known design patterns exclusive from (1) to assure the balance of learning accuracies between training data and unknown testing data, especially in Step B. These models can then be applied generically to any other designs to classify datapath clusters.

4.2 Cluster Classification and Evaluation

Once Step A is completed, in Step B the data learning models will be applied directly to classify and evaluate new unknown design patterns. As the new patterns go through the learning models, the evaluation scores could span within certain range for datapath and non-datapath patterns respectively for NN and SVM. In this step, we evaluate a pattern to be datapath like if and only if both NN and SVM evaluation scores are above certain thresholds. This helps to systematically improve the datapath evaluation accuracy without noticeable penalty in non-datapath accuracy.

Usually NN and SVM have similar performance for most of binary classifications, e.g., differentiating datapath-like and non-datapath patterns. In principle, SVM guarantees the global optimum but is sensitive to data noise. NN usually has good noise-robustness, however it takes more time in the training and calibration step to reach optimal or close-to-optimal. Each C_i identified as datapath logic is passed to the bit-stack assignment in the next section.

5. BIT-STACK SELECTION WITH ILP

Once a cluster has been classified as containing datapath logic, step 5 from Fig. 2 extracts the bit-stack structures from the logic clusters and passes those bit-stacks to the datapath placer. For each cluster C_i, a set of bit-stack candidates is chosen based on maximizing the total bit-stack count. This work uses the automorphism generators as the bit-stack candidates and adds in wire-length weighting to make the ILP formulation wire-length aware.

5.1 Bit-Stack Candidate List

Each generator set S_i, created during classification of each C_i, captures possible cell connections that can be used for a bit-stack assignment. Figure 3(a) provides an example for clarity. The generator group for the graph in Figure 3(a) is: $(1,2)(3,4)$, $(5,6)$ and $(1,5)(2,6)$. Using this generator set, it is possible to create bit-stack candidates by grouping the tuple by index-0 and index-1 from each generator. For Fig. 3(a), the bit-stack candidates would be:

$$b_0 = [1:3:5] \qquad b_2 = [1:2]$$
$$b_1 = [2:4:6] \qquad b_3 = [5:6]$$

Thus, with a set of bit-stack candidates, the goal is to maximize the number of bit-stacks within the partition while maintaining mutual exclusion among the cells. This constraint maintains the requirement that a particular cell can not be assigned to multiple bit-stacks.

5.2 ILP-based Bit-Stack Selection

The bit-stack candidate selection is optimally solved using integer linear programming (ILP). A binary vector η is maximized with the linear function $\Gamma^{\mathbf{T}}(\eta)$ subject to the non-overlap constraint. Assuming there are $i = 1...n$ bit-stack candidates, let η be a binary indicator variable such that:

$$\eta_i = \begin{cases} 1 & \text{if bit-stack candidate } B_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Let $\alpha_i = |B_i|$ and $\beta_i = w_i$ where w_i is equal to the Half-Perimeter Wire Length of the edges connected to each cell in bit-stack candidate B_i. The α_i term increases the value for larger bit-stack candidates (covering more cells) and the B_i term adds a penalty for larger wire-length. Then the objective function maximizes the number of bit-stack candidates selected as given in Equation 3.

$$\underset{\eta}{\text{maximize:}} \quad \Gamma = \sum_{i=0}^{n} [(\frac{\alpha_i}{\beta_i} * \eta_i)] \tag{3}$$

$$\begin{aligned} \text{subject to:} \quad & \eta_i + \eta_j \leq 1, && \forall\, i,j \iff \eta_i \cap \eta_j \\ & 0 \leq i < j < n && \forall\, i,j \\ & \eta_i \in (0,1) && \forall\, i,j \end{aligned} \tag{4}$$

Equation 4 maintains the non-overlapping cell constraint $\eta_i \cap \eta_j = \emptyset$ between each bit-stack candidate. Though the general ILP problem is NP-Hard [16] and the solution time for the integer programming problem grows exponentially (in the worst case) with the number of integer variables, in this case the run time is negligible for two reasons: (1) The number of bit-stack candidates n from a single C_i and the number of constraints is generally very low, with n often on the order of a few hundred because we bound the size of C_i; (2) The ILP assignment only occurs when a cluster is classified as datapath logic meaning for the majority of the clusters, the ILP code does not run at all.

With the preceding steps from Fig. 2, PADE is able to quickly extract and classify datapath structures then pass them to an ILP solver to generate the bit-stacks for the logic. By making the classification and bit-stack assignment aware of physical placement information from the global placer, significant improvement in overall wirelength is possible as will be shown in the next section.

6. EXPERIMENTAL RESULTS

The first step, at a one-time cost, was training the high-dimensional models using known datapath pattern extracted from four baseline industrial hybrid circuits and the ISPD2011 Datapath Benchmark Suite [5]. Both datapath and random logic patterns were trained off the baseline circuits. Then the global placement flow was developed to extract and evaluate each C_i within the original netlist using the high-dimensional model. Clusters identified as containing datapath structures were mapped to bit-stacks using the automorphisms of the subcircuit and the ILP formulation. After the bit-stack was defined, global placement continued through completion and then detailed placement and legalization ran. All numbers reported are total wirelength results for both datapath and random logics on legal placement solutions.

PADE was implemented in C++ with g++ 4.1.2 extending the SAPT [6] placer for automatic datapath extraction. Running PADE without datapath extraction results in the same wirelength results reported for SimPL because SAPT extended the SimPL framework. Benchmark runs were performed on an Intel Xeon CPU x5570 Linux workstation running at 2.93GHz using two CPU cores. This work compared PADE against six *untrained* industrial hybrid designs and additionally on the *untrained* ISPD 2005 benchmark suite [2]. For improved experimental control, all HPWL numbers and StWL estimates were generated using coalesCgrip [17][3], every placer was run in *default mode*, and all placers were supplied a target density requirement of 1 as defined as in ISPD placement contests [2]. The tool bliss [14] was used to generate the automorphism groups for each cluster and GUROBI [18] for the lp solver. Wire-length results for the ISPD 2011 Datapath Benchmark circuits are not provided because they were used to train the high-dimensional models.

6.1 High Dimensional Learning Accuracies

We implemented and fine-tuned both SVM and NN algorithms specifically for the evaluation of datapath patterns. Then we combine both of their evaluation scores for datapath extraction. The NN accuracy is shown in Fig. 5 and SVM accuracy is shown in Fig. 6

A 2 class C-SVM algorithm is modified and configured at a one time training and calibration cost of around 3 minutes, involving: (1) training/calibrating of SVM models over some known structures with around 100 datapath and 10K non-datapath; (2) validation of the calibrated models over a relatively large set of known datapath/non-datapath structures beyond (1) with around 300 datapath and 60K non-datapath. Step(1) shows about 85% and 99.5% of datapath and non-datapath accuracy respectively, while Step(2) reaches 80% and 99% of datapath and non-datapath accuracy respectively. In the calibration and scoring process, a separation threshold of -0.9 is used for SVM models. A resilient backward propagation NN algorithm is fine-tuned within around 8 minutes using similar steps. It shows 90% and 99.9% of datapath and non-datapath accuracy in training, 87.8% and 99.6% in validation, with a separation threshold 0.05.

6.2 Wire Length Results

In the tables that follow, PADE refers to the proposed placement technique with automatic datapath extraction and evaluation. To compare the data learning and extraction effectiveness of PADE, we also implemented *Logic Based Regularity Extraction* (LBRE) based on [22]. Everything in LBRE is the same except the extraction and bit-stack assignment techniques. The logic based regularity extraction results are passed to the same datapath placer and compares the effectiveness of prior extraction techniques verses PADE. LBRE uses functional regularity to extract the datapath therefore can only be compared against the hybrid circuit designs because logical information is not provided in the ISPD 2005 benchmark circuits.

[3]FastPlace3 [7] reports slightly lower HPWL than CoalesCgrip.

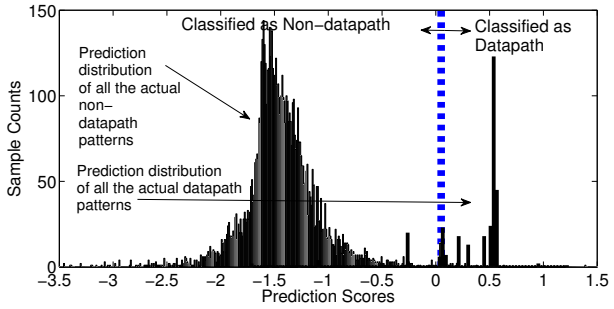

Figure 5: Validation accuracies of datapath and non-datapath by NN on relatively large data set

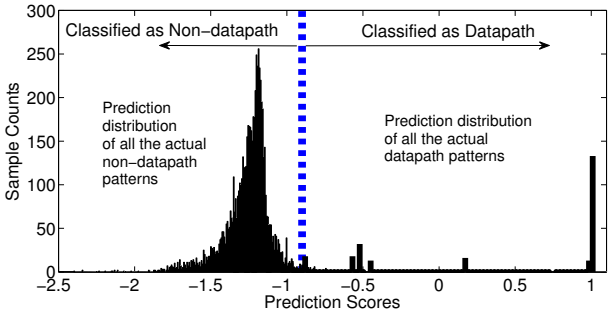

Figure 6: Validation accuracies of datapath and non-datapath by SVM on relatively large data set

Wirelength results on six industrial hybrid circuits and the ISPD 2005 Placement Benchmarks are presented in Table 1. Each of the hybrid designs are state-of-the-art circuits containing a mixture of random and datapath logic. Though the exact ratio of datapath to random logic is not known, generally the significant majority of the logic is random. As discussed in [6], HPWL to StWL correlation can be inadequate for datapath logic. Thus, both HPWL and StWL is reported with the best StWL result in bold. In every case, PADE obtains the best StWL results. In four of the six cases, PADE also outperforms all other placers in HPWL results.

The purpose of running on the ISPD 2005 placement benchmark suite is to show that the methods herein are capable of high placement quality on both random and datapath logics. Surprisingly, some datapath structure was found and on average PADE improves the HPWL 2% and StWL by 3% compared to prior academic placers. As Table 1 shows, PADE produced the best HWPL and StWL results for seven of the eight benchmarks.

One notable placer missing from our comparisons is the structure aware Beacon placer [10]. Though requested, currently the placer does not work with mixed-size placement and thus direct comparison is not possible.

6.3 Runtime Comparisons

Table 2 compares the runtime of PADE against other state-of-the-art placers. For the hybrid and ISPD 2005 Benchmark circuits, FastPlace3.1 ran the fastest of all placers. For the hybrid circuits, PADE was only 19% slower than FastPlace3.1 and for the ISPD 2005 benchmarks, PADE was 32% slower than FastPlace3.1. Overall, PADE significantly outperforms CAPO10.2, mPL6, and NTUPlace3 showing speedups of 7.28, 3.26 and 1.74 respectively on the ISPD 2005 Benchmarks. Though PADE is not the fastest placer, there is clearly a significant wirelength benefit on hybrid design styles and it is possible to parallelize the clustering, evaluation and

Circuit	Capo10.5 [19]		mPL6 [20]		FastPlace3.1 [7]		NTUPlace3 [21]		simPL [1]		LBRE		PADE	
	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL
Hybrid 1	2.39	3.12	2.27	2.89	2.06	2.61	2.04	2.67	2.19	2.75	2.32	2.89	2.05	**2.55**
Hybrid 2	1.72	2.51	1.47	2.17	1.39	2.20	1.38	2.20	1.39	2.20	1.39	2.18	1.37	**1.87**
Hybrid 3	2.68	2.75	1.89	2.41	1.81	2.28	1.77	2.25	1.77	2.25	1.79	2.26	1.71	**2.16**
Hybrid 4	2.66	3.57	3.18	4.01	2.91	3.59	2.35	3.36	2.69	3.30	2.79	3.49	2.36	**2.77**
Hybrid 5	11.36	12.90	12.76	14.41	10.88	13.27	10.59	12.30	10.57	12.22	10.56	12.22	9.71	**10.87**
Hybrid 6	7.66	9.06	9.04	10.29	7.75	9.04	9.04	10.69	6.64	7.92	6.90	8.21	6.24	**7.21**
Average	**1.25**	**1.30**	**1.23**	**1.27**	**1.11**	**1.17**	**1.09**	**1.18**	**1.07**	**1.12**	**1.10**	**1.14**	**1.00**	**1.00**
Adaptec1	88.14	97.22	77.58	86.20	79.88	88.75	81.82	91.06	78.15	87.05	-	-	76.83	**85.12**
Adaptec2	100.25	114.54	90.31	100.64	93.02	104.03	88.79	99.06	90.96	102.13	-	-	89.14	**98.92**
Adaptec3	276.80	296.22	215.88	235.06	219.78	239.70	214.83	234.52	208.81	228.32	-	-	205.32	**222.08**
Adaptec4	231.30	257.47	193.93	208.85	199.66	215.02	195.93	211.86	187.21	201.82	-	-	183.79	**196.23**
Bigblue1	110.92	127.72	97.10	108.31	94.37	**105.24**	98.41	110.02	98.64	109.94	-	-	95.86	106.98
Bigblue2	162.81	189.60	152.13	174.69	155.16	178.44	151.55	175.27	145.29	168.65	-	-	143.18	**164.33**
Bigblue3	405.40	452.91	342.50	370.70	392.72	421.31	360.66	389.39	341.55	369.61	-	-	341.72	**361.96**
Bigblue4	1016.19	1105.52	831.34	930.63	816.14	911.64	866.43	974.44	804.22	901.85	-	-	796.18	**883.82**
Average	**1.21**	**1.22**	**1.03**	**1.04**	**1.06**	**1.07**	**1.05**	**1.06**	**1.02**	**1.03**	**-**	**-**	**1.00**	**1.00**

Table 1: Legal HPWL and StWL (x10e6) comparison on industrial hybrid designs and the ISPD 2005 Placement Benchmarks [2]. HPWL and StWL was computed using CoalesCgrip [17]. LBRE is blank for the ISPD 2005 suite because logic information is not provided for those circuits.

	Capo10.2	mPL6	FP3.1	NTUPlace3	simPL	LBRE	PADE
Hd1	7.4	1.9	1.3	1.5	1.2	2.1	1.1
Hd2	8.1	2.4	1.4	1.7	2.0	2.7	1.7
Hd3	8.8	4.5	1.7	2.7	2.2	4.2	1.8
Hd4	8.6	4.1	2.2	3.2	2.2	6.3	2.3
Hd5	25.2	10.7	6.4	9.8	5.8	12.8	7.4
Hd6	45.4	22.8	4.8	9.3	4.1	14.2	7.1
Ave	**4.99**	**2.01**	**0.81**	**1.31**	**0.97**	**2.05**	**1.00**
ad1	35.7	18.3	4.7	10.0	4.2	-	5.3
ad2	42.8	19.9	2.2	9.2	4.4	-	5.6
ad3	111.9	60.3	4.4	18.6	10.7	-	12.9
ad4	110.0	58.5	9.0	19.5	18.1	-	21.4
bb1	56.6	21.8	5.4	16.2	4.5	-	5.5
bb2	107.6	64.0	9.6	32.1	17.3	-	20.4
bb3	286.0	88.4	28.3	62.5	34.8	-	40.6
bb4	543.4	172.8	58.1	141.8	62.3	-	73.0
Ave	**7.28**	**3.26**	**0.68**	**1.74**	**0.83**	**-**	**1.00**

Table 2: The total runtime comparisons (sec). Runtimes on the ISPD 2005 benchmarks on LBRE are left blank because logical information is not provided by the ISPD 2005 benchmarks. (hd = hybrid, ad = adaptec, bb = bigblue, FP3.1 = FastPlace3.1)

bit-stack assignment stages of the flow. Doing so would reduce runtimes to be similar with the other state-of-the-art placement algorithms.

7. CONCLUSIONS

This work presented a high-performance mixed-size placer PADE with automatic datapath extraction and evaluation through high-dimensional data learning using both logical and physical information. PADE has demonstrated 7% improvements in HPWL and 12% improvements in StWL for a set of industrial hybrid circuits compared to prior placers. Even for the ISPD 2005 benchmark circuits, PADE produces 2% average improvements for HPWL and 3% improvement in StWL over prior placers. To our best knowledge, this is the first attempt that links high-dimensional data learning with placement of hybrid datapath and random logic circuits. The results are very encouraging and we believe a lot of future research can be done to further advance the state-of-the-art of modern placement.

8. REFERENCES

[1] M.-C. Kim, D.-J. Lee, and I. L. Markov, "simPL: an effective placement algorithm," in *Proc. ICCAD*, pp. 649–656, 2010.

[2] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz., "ISPD 2005 placement contest benchmark suite," in *Proc. ISPD*, pp. 216–220, 2005.

[3] G.-J. Nam and J. Cong, eds., *Modern Circuit Placement: Best Practices and Results*. New York, NY: Springer, 2007.

[4] R. X. T. Nijssen and J. A. G. Jess, "Two-dimensional datapath regularity extraction," in *IFIP Workshop on Logic and Architecture Synthesis*, pp. 110–117, 1996.

[5] S. I. Ward, D. A. Papa, Z. Li, C. N. Sze, C. J. Alpert, and E. Swartzlander, "Quantifying academic placer performance on custom designs," in *Proc. ISPD*, pp. 91–98, 2011.

[6] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. Alpert, E. Swartzlander, , and D. Z. Pan, "Keep it straight: Teaching placement how to better handle designs with datapaths," in *Proc. ISPD*, 2012.

[7] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proceedings of ASPDAC*, pp. 135–140, 2007.

[8] A. Chowdhary, S. Kale, P. Saripella, N. Sehgal, and R. Gupta, "Extraction of functional regularity in datapath circuits," *IEEE TCAD*, vol. 18, no. 9, pp. 1279–1296, 1999.

[9] A. Rosiello, F. Ferrandi, D. Pandini, and D. Sciuto, "A hash-based approach for functional regularity extraction during logic synthesis," in *Proc. ISVLSI*, pp. 92–97, 2007.

[10] S. Ono and P. H. Madden, "On structure and suboptimality in placement," in *Proceedings of ASPDAC*, pp. 331–336, 2005.

[11] P. Ienne and A. GrieBing, "Practical experiences with standard-cell based datapath design tools," in *Proceedings of DAC*, pp. 396–401, 1998.

[12] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net model," *IEEE TCAD*, vol. 27, no. 8, pp. 1398–1411, 2008.

[13] Q. Liu and M. Marek-Sadowska, "Pre-layout physical connectivity predictions with applications in clustering, placement and logic synthesis," in *Proc. ICCAD*, pp. 31–37, 2005.

[14] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," 2007.

[15] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working Set Selection Using Second Order Information for Training Support Vector Machines," in *Journal of Machine Learning Research*, 2005.

[16] J. E. Beasley, ed., *Advances in Linear and Integer Programming*. New York, NY: Oxford University Press, Inc., 1996.

[17] H. Shojaei, A. Davoodi, and J. Linderoth, "Congestion analysis for global routing via integer programming," in *Proc. ICCAD*, pp. 256–262, 2011.

[18] G. Optimization, "The gurobi optimizer 4.5." `http://www.gurobi.com/`.

[19] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: robust and scalable open-source min-cut floorplacer," in *Proc. ISPD*, pp. 224–226, 2005.

[20] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: enhanced multilevel mixed-size placement," in *Proc. ISPD*, pp. 212–214, 2006.

[21] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "A high-quality mixed-size analytical placer considering preplaced blocks and density constraints," in *Proc. ICCAD*, pp. 187–192, 2006.

[22] T. Kutzschebauch and L. Stok, "Regularity driven logic synthesis," in *Proc. ICCAD*, pp. 439–446, 2000.