Boosting Convergence of Timing Closure using Feature Selection in a Learning-driven Approach

Que Yanghua
School of Computer Science and Engineering
Nanyang Technological University
Singapore, 639798
yanghua.que@ntu.edu.sg

Harnhua Ng
Plunify Inc.
82 Geylang Lor 23,
Singapore, 388409
harnhua@plunify.com

Nachiket Kapre
School of Computer Science and Engineering
Nanyang Technological University
Singapore, 639798
nachiket@ieee.org

***Abstract*—**
Machine Learning approaches for automated selection of FPGA CAD tool parameters have been demonstrated to be useful for timing closure of FPGA designs [3], [4]. This is achieved by running the CAD tool multiple times with small variations in the the CAD parameter values. The timing slack from each run is recorded into a database along with all input parameter selections to help train a classifier. By progressively running more instances of the tool, we can help drive the CAD tool towards timing convergence. However, a naïve approach that uses simplistic off-the-shelf learning libraries and uses all features (CAD parameters) is inappropriate. This can often miss opportunities inherent in specific design properties and nuances of the FPGA device family and tool versions while possibly overfitting the models and trapping the system into a local minima. In this paper, we show how to combine design-specific feature selection with a set of classification approaches that are configured to improve model quality and reduce the number of iterations required to deliver timing closure. We show how to systematically tailor the correct subset of features for each design to deliver robust results. Using design-specific feature selection, we prune the set of CAD parameters used for constructing the classifier model down from ≈80 to ≈8–22 features. We show improved AUC scores (Area under ROC curve) as high as 0.83 which represents an improvement over the baseline InTime scores of 0.74 earlier. We use a set of large industrial designs to show these results and lower the number of CAD iterations required for convergence by 3× (mean) using our proposed approach.

I. Introduction

FPGA CAD tools navigate the large optimization space of mapping RTL designs to the FPGA fabric through extensive use of tuneable heuristics. As each step of the CAD mapping flow *i.e.* synthesis, packing, placement, and routing is typically NP-complete, the use of heuristics is necessary to generate FPGA bitstreams in a reasonable amount of time. Modern FPGAs support millions of LUTs, thousands of DSPs and on-chip Block RAMs, along with various exotic hardware features, resulting in compilation times have risen to hours or days. When dealing with timing constraints in the design flow, the FPGA CAD tools are often used iteratively to deliver timing closure through and edit-compile-debug approach further exacerbating the challenge. Modern releases of the FPGA CAD tools now expose hundreds of tuning parameters as hints for helping experienced developers drive the heuristics on large designs with tough constraints. The combination of long runtimes and complex configuration options make it increasingly difficult for a human developer to manage this design process.

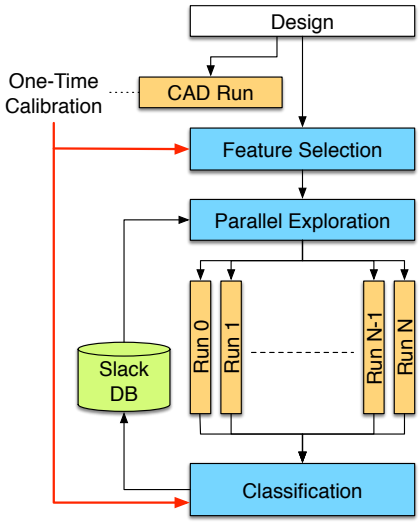


Fig. 1: Organization of InTime Flow.

In this study, we use InTime [3], [4], a plugin for FPGA CAD tools, that can automatically select tool parameter assignments for each design through the effective use of machine learning heuristics and cheap cloud computing resources. While a modern CAD tool exposes hundreds of parameters, InTime identifies the set of 25 parameters that may be relevant based on design-specific ranking. It then iteratively attempts a series of parallel CAD runs with different CAD parameters combinations that lead to timing closure. Considering boolean parameters, this still represents a search space of $\approx 2^{25}$ combinations (down from a possible 2^{100s} candidates).

This tool has been demonstrated to work successfully on large industrial problems. For instance, one particular tough problem handled by InTime was a UDP offload accelerator supporting 2×100GbE L2 frames and 1×25GbE UDP frames at 390 MHz clock. This was a large design mapped to a modern Xilinx Ultrascale `xcvu095` device with 25% logic utilization

(chip capacity: 500K LUTs and 1M FFs). The design was already deeply pipelined and highly optimized to manage the strict 390 MHz timing constraint. Despite their best effort and availability of spare resources (75% free), developers were unable to tackle the timing constraint solely through pipelining and were left with 100ps of negative slack. Each compilation took 8–9 hours on wall clock time when using Vivado 2015.2 CAD tool on a modern 16-core 64b CentOS 6.5 platform. With the use of InTime, it was possible to deliver timing closure in a fully-automated manner in under 2–3 days of exploration.

In this paper, we focus on reducing the number of iterations required to achieve timing closure. This requires indirectly exploiting design-specific properties *i.e.* richness of DSP use, path delay profiles, extent of pipelining to guide the **feature selection** process. Feature selection helps reduce the complexity of the predictive model used to evaluate goodness of a particular combination of CAD parameters. Typically, fewer features lead to poor prediction accuracy while excess features lead of overfitting and resulting accuracy drop. We further use a previously developed robust classifier [13] by exploring multiple learning algorithms that work well with limited number of data samples. FPGA CAD runtimes are long, and even when cloud computing resources are abundantly available, it is prudent to spend those resources efficiently on useful work.

The key contributions of this report include:

- Quantification of the necessity of machine learning for delivering timing closure in FPGA designs on complex modern FPGA CAD tools.
- Development of a **feature selection** approach that helps customize the subset of features for each design to help improve accuracy of the learning routines.
- Quantification and characterization of these optimizations across various industry-strength benchmarks in terms of machine learning metrics (*e.g.* ROC, AUC) and FPGA outcomes (*e.g.* TNS trends, iteration counts).

II. Background

As shown in Fig. 1, InTime is an iterative algorithm organized as a series of concurrent CAD *runs*. Each *round*, which consists of multiple concurrent *runs*, is an opportunity to generate candidate CAD parameter combinations and acquire data for analysis. Within each round, InTime uses a supervised learning approach to train classifiers that evaluate the effectiveness of a given combination of CAD parameter selections towards reducing timing slack. The baseline reference used for training the classifier is derived from the *default configuration* of the FPGA CAD tools with unmodified vendor-provided presets. This can be formalized as shown in Fig. 2.

- n is the total number of CAD tool parameters. *i.e.* Quartus 14.1 exports ≈80 parameters
- p is the number of CAD tool runs in a given round
- i is a CAD tool parameter $1 \leq i < n$.
- j is an instance of running a CAD tool $1 \leq j < p$.
- The InTime trials generate a matrix X that stores the assignments for the different CAD tool parameters, where x_{ij} corresponds to each CAD tool parameter and indicates if that parameter is set. *e.g.* if $x_{ij} = T$, the parameter i is set during the CAD tool execution j, and vice versa.
- Vector Y records the resulting timing slack results y_j from p parallel trials of the CAD tool.
- InTime divides the set of experiments p into multiple *rounds* r to ensure the search space is covered in a design-specific, adaptive manner.

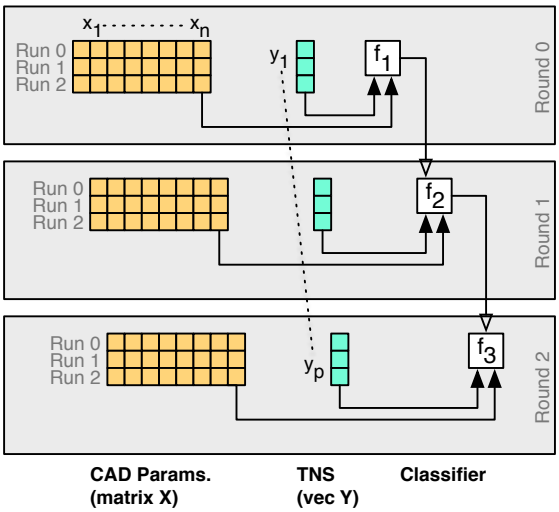


Fig. 2: Various steps of building and improving the classifier f_r in each learning round r. Multiple parallel CAD iterations are organized into *rounds*. Results from a round are collected together and used to updated the classifier function f_r.

One may be tempted to use this data X and Y to build a predictive model for directly estimating the value of timing slack for a new, untested combination of CAD tool parameters. Unlike modern high-level synthesis, packing and placement tools that use a predictive delay (and resource) model based on a *continuous outcome predictor*, InTime reduces this to a mere *classification* problem where we identify whether each y_j timing score is either `GOOD` or `BAD`. This leads to a supervised learning approach; we compare the timing slack of a given execution against a baseline default run of the CAD tool to judge the goodness of a given combination. This classification function f, shown in Fig. 2, is the one that can explain the dataset presented and is periodically refined when new data is added to the tables. The challenge for InTime is to train this classifier model for (1) high accuracy and low false positive rates, (2) requiring the fewest possible set of samples p (thus, moderating the use of compute costs required to deliver this result).

For a concrete example of InTime in action, consider the UDP offload accelerator design we introduced earlier. We show the improvement in WNS scores over time in Figure 3. We are able to lower the WNS scores from 300 ps down to 83 ps within a few days using machine learning with eventual timing closure delivered through a seed exploration in another day. InTime started optimizing the design without any prior

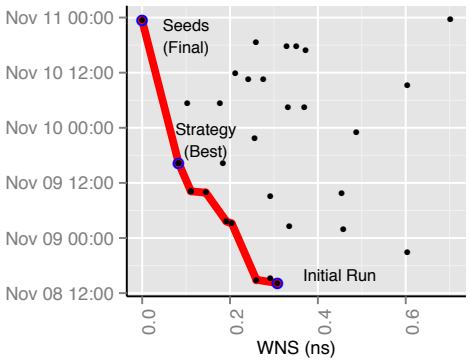


Fig. 3: Worst-Case Negative Slack improvements vs. time for UDP offload engine. *Strategy* is the machine-learning driven improvement in WNS scores, *Seeds* if the final seed exploration to mop up remaining slack.

knowledge and relied on internal calibration data for the target device and CAD tool version. From the developer's perspective, they launch InTime with the design RTL along with timing constraints and let the tool handle the rest of the flow. Thus, the complete flow is fully automated, with InTime managing the machine learning rounds as well as the seed exploration to deliver timing closure. As we can see in the Figure, the machine learning approach is used to bring WNS slack withing a threshold first. The final timing slack is recovered without relying on machine learning through seed exploration rounds called multiple cost tables in ISE which are parallel, or iterative Vivado runs which are sequential[1]. In this particular case, we use Vivado iterative placement loop for three turns to deliver the final timing closure from the 83 ps design as the starting point.

III. Configuring Machine Learning

InTime originally used the simplistic Naïve Bayesian classifier (`nb`) to drive the learning process and rank the relative importance of the CAD tool parameters while filtering the most important ones using Principal Component Analysis (PCA). The Naïve Bayes classifier operates by building a probability vector Pr where p_i is the probability that the ith CAD parameter is set. We refine these probabilities for each observation (CAD run) by determining when the timing score is better than the threshold. Once built in a given *round*, we can then generate suitable candidate CAD parameter combinations by testing against this model. Using this approach, InTime is able to reduce timing scores by 10× [3] in under 200 runs of the CAD tool. While these results were promising, our work improves the original InTime approach in the following ways:

- **Learning Algorithm**: A variety of mature classifier algorithms are available that work well under different circumstances (quality, quantity) of the input dataset. In this paper, we explore a suitable set of candidate algorithms suitable for FPGA CAD datasets. Our dataset sizes are fairly small (*i.e.* 200-400 samples per design, compared to other millions of records in big-data applications) due to the long runtimes of the CAD tools and finite computing costs at our disposal. What makes this even more challenging is the finicky nature of the resulting timing slacks to even the slightest change in input CAD parameter conditions.
- **Feature Selection**: While InTime relied solely on PCA analysis to identify useful features, we exploit design knowledge of the particular problem to help choose required parameters. This approach helps us properly tune the number of important parameters as desirable for optimizing accuracy metrics while simultaneously reducing the number of CAD parameter combinations that are considered productive for search.

We move beyond the simplistic approach of original InTime and consider (1) design-specific features selection, coupled with (2) design-specific classification heuristics [13] to deliver faster timing closure.

A. Feature Selection

Feature selection is a process that aims to reduce the number of features (CAD parameters in our application) used for model construction. It decreases the chance of overfitting and delivers boosted performance as well as faster time to convergence. We consider the importance of each feature individually and use statistical metrics to quantify magnitude of change in classifier performance when that particular feature is used. It is somewhat analogous to sensitivity analysis in design engineering. The intuition here is that each FPGA design has unique timing and device utilization characteristics that interact differently with the CAD heuristics. Thus, instead of simply using all available CAD parameters for building the predictive model, we prune the set of features to only consider those that align with design properties. For instance, a DSP-rich design would benefit from DSP-centric technology mapping heuristics in the CAD tool. The exact subset of features that may matter varies with the design and we discover that in most cases, ≈10–20 features are necessary for boosting classifier performance. However, it is important to note that the combinations that we explore may still modify the less important parameter values. This coverage is useful to avoid the model getting stuck in a local minima during the parallel search. The primary advantage we gain through parameter pruning is to boost predictor accuracy and focus the search in a productive space of CAD parameter combinations. We enumerate the various feature selection functions we evaluation in Table I and briefly describe how they work as follows:

- **One Rule**: (R function `oneR`) This feature selector creates association rules by identifying the correlation between a particular feature and its impact on the output class. It builds a simple frequency table and ranks the various features based on their impact on the final classification outcome.

[1]The regression to an iterative, sequential approach is a step back for Vivado if speed to timing closure via parallelization of seed explorations is desirable.

TABLE I: Feature Selection Equations.

Selection	Formula
OneR	$max(class)$
Information Gain	$H(Class) + H(Feature) - H(Class\|Feature)$
Relief	$S = \frac{1}{2}\sum_{k=1}^{l} d(X_k - X_{M_k}) - d(X_k - X_{H_k})$

After we run the initial round of multiple CAD runs, the resulting data is used to extract the parameter list.

- **Information Gain**: (R function `information.gain`) Here we use a more sophisticated approach for ranking features by computing entropy of each feature. Instead of simply counting the associations of a feature with classification outcome, we track the amount of information (new knowledge) that is gained by inclusion of a particular feature. Intuitively, it calculates the number of bits of information that is learned about the behavior of the feature when its selected to train a classifier model. The more bits a feature contributes, the higher its relative importance.
- **Relief:** (R function `relief`) This is well-suited for binary classification problems like the one explored in this paper. It operates by evaluating a distance $d()$ score for each CAD run with other runs with fewest variations in the CAD parameter selections. This is then used to calculate a relevance score.
- **Ensemble:** (R function `ensemble`) As no single feature selection algorithm is likely to be optimal in all cases [9], we also consider an ensemble approach by combining multiple selectors and computing a mean score for each feature. As we show later in Section IV (Figure 8), the `ensemble`-based feature selector outperforms the other three individual approaches listed above.

Classification is the procedure used to determine whether a given combination of CAD parameters yields a `GOOD` timing score. A challenge for most classifiers is to operate in presence of limited samples (30 iterations per round). Once important features have been suitably ranked and used to train a model, we must consider the interaction of design characteristics with properties of the classification algorithm for delivering robust performance. In [13], we evaluate a variety of routines such as **Logistic regression** (`glm`), **Bagging** (`treebag`), **Random Forest** (`rf`), **Support Vector Machine (SVM)** (`svmRadial`), **Neural Network** (`nnet`) along with **Stack** and **Ensemble** methods. As we will see later, the exact choice of the classifier is not nearly as important as the feature-based ranking and selection of required parameters.

IV. Results

In this section, we present quantitative results of our experiments and briefly explain the experimental setup. We develop the feature selection and machine learning routines as plugins for InTime using *caret* [1] and *FSelector* [7] packages in `R` [6]. The use of these APIs was cross-validated and optimized for high learning performance while compute times are insignificant compared to CAD tools runtimes. We do not expect the end users of InTime to use this interface

TABLE II: Characteristics of Industrial Designs[1].

Name	LUTs (% used)	FFs	P&R mins.	TNS	WNS
office_jpn2	78	107K	100	26.9	7.3
autom_jpn	61	108K	136	67.3	7.9
net_chn3	73	59K	121	69.1	5.1
net_isr1	81	160K	225	3.6	10.5
net_isr2	79	158K	194	4.2	8.9
net_isr3	75	372K	438		4.7
net_chn4	91	14K	44	23.2	13.9
SOC	8	4K	31	23.1	3.4
VIP	67	63K	70	3.1	2.8

[1]Exact name of design and some details obfuscated to ensure design privacy.

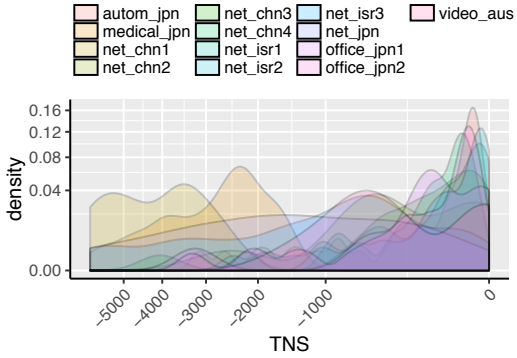


Fig. 4: Histogram distribution of TNS scores for various benchmarks. Higher density of distribution close to zero suggests plenty of opportunity to drive the solution to timing closure.

and it is only supported for developers of InTime. Within the InTime flow, we rerun feature selection at each stage of the model refinment (*i.e.* after each *round*). This means, that the exact set of features used in the revised model may be different for each *round*. We ran our experiments on a combination of resources including the Google Compute Engine and internal compute farms. Typically each design ran for a few days before meeting timing or aborting. A small number of industrial designs with light pipelining are unable to meet timing as expected due to abnormally high TNS and WNS scores. A particularly challenging design was the 390 MHz UDP offload accelerator that was already deeply pipelined and provided very little slack to start with. Most of our designs are generally of this nature. In Table II, we list the key characteristics of the industrial benchmarks used in this study. They cover a wide range of application scenarios such as wired networking, image processing, as well as SoC designs. These benchmarks are compiled using a combination of Quartus 14.1 and Vivado 2015.2 and mapped to the a range of large Ultrascale Virtex and Stratix V devices. InTime uses the *default* settings in these CAD tools to extract the initial baseline for use in the classification process. We show a density distribution of achieved TNS scores for the various designs in Figure 4. This indicates a spread in achievable TNS scores and in some cases even the opportunity for exceeding the timing target. Most designs have a clear peak which is

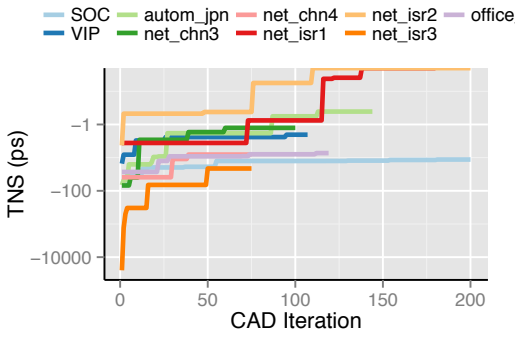


Fig. 5: TNS convergence of various benchmarks with InTime (logscale y-axis, most designs with non-zero TNS scores still converge in a final seed exploration cleanup phase).

close to the average performance of the design under nominal CAD parameter settings. Since our machine learning approach is adaptive, the precise starting condition does not matter to eventual convergence. For these benchmarks, in Figure 5, we observe how InTime consistently reduces total negative slack (TNS) close to zero within 120 iterations. A few benchmarks did not meet timing – (1) The `net_isr3` design started with an unrealistic initial TNS score of 10000 ps and did not converge. That design has to be pipelined adequately prior to using InTime. (2) The `SOC` design ran the longest and showed practically no improvements in TNS scores. This design exhibited stubborn timing paths that are constrained by the limits of the device than FPGA CAD margins.

We now define important metrics used to evaluate the efficacy of our machine learning framework:

- **ROC**:(Receiver Operating Characteristics): Visually, we can compare various approaches with the ROC curve that plots TPR (True Positive Rate) against FPR (False positive Rate). Here, we want to be in the upper left quadrant of the curve with high TPR and low FPR values.
- **Area Under Curve (AUC)**: AUC is the area under the ROC curve and provides a fair estimation of classification accuracy by discounting rate of misinterpretation. A high AUC value indicates that a model with high true positive rate and low false positive is achievable. AUC of 1 is perfect and AUC of 0.5 is effectively an unbiased random coin toss.
- **Entropy**: Mathematically, entropy is defined as $S = \sum_i p_i \cdot log_2(p_i)$ where i is the CAD parameter, and p_i is the probability that a given parameter is set. High values of entropy indicates a greater extent of randomness in the system. An entropy value of 0 indicates that a single set of values are assigned to the parameters in all CAD runs.

A. Necessity of Learning

In Figure 6, we show the entropy of the system when considering the set of experiments covered and the resulting solution values. High values of entropy suggests that InTime covered a wide range of *distinct* parameter combinations during its search. When considering the entropy of the combinations that

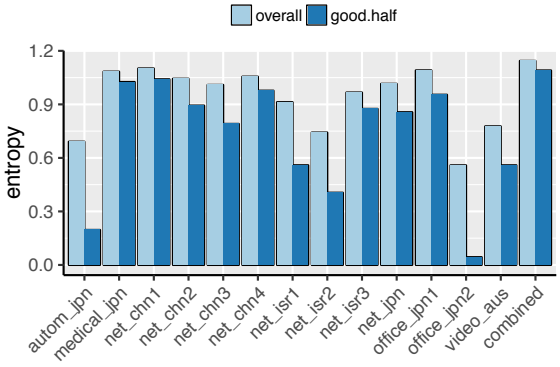


Fig. 6: Entropy of the explored CAD parameter combinations and the dissimilarity of optimal parameter combinations needed for timing convergence (column `combined`). `overall` corresponds to all CAD runs, while `good.half` corresponds to the TNS-improving combinations.

.

delivered the best TNS scores (*i.e.* timing convergence), the resulting entropy is still quite high, see `combined` bar in the plot. This confirms our hypothesis that the optimal solutions have sufficient divergence and cannot be trivially clustered into small set of bins. Thus, machine learning is crucial to delivering timing convergence as we need to tailor the search per design and identify the unique parameter combination that are needed by that design.

To further cement this understanding, we evaluate the degree of similarity between the final parameter combinations for the various designs in Figure 7. Here, we plot a matrix (heatplot) of scores that show the extent of dissimilarity between the final parameter combinations for various designs. The more dissimilar the final parameter combinations, the corresponding box will be darker (diagonal is white as its perfectly self-similar). To plot this figure, we extract the combinations per design that yield the best TNS scores (timing closure) and compute the overlap between these choices across designs. We pairwise compare the designs and visualize the resulting scores in the heatplot shown. As it clear, barring a few cases we observe 40–60% dissimilarity in most scenarios. This means that 40–60% of the parameter values were differently chosen to deliver timing closure for the two designs. Thus, the final results have no clear correlations and coupled with the higher entropies observed in Figure 6, make it evident that we need design-specific customization and machine learning to drive the search.

B. Understanding Effectiveness of Feature Selection

Unlike the original InTime approach, we use design-specific feature selection as a first step, prior to training, to help improve the quality of the prediction. In Figure 8, we show the overall AUC scores achieved by performing intelligent feature selection on the `medical_jpn` benchmark before

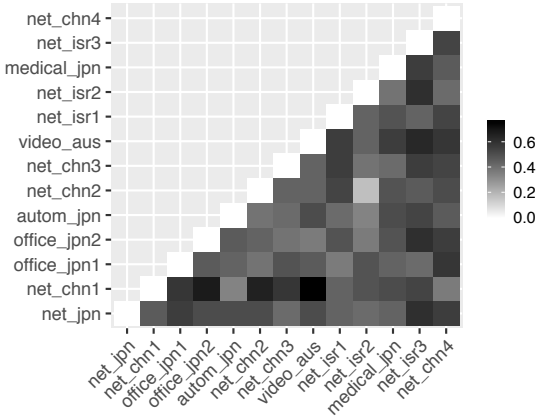


Fig. 7: Dissimilarity of the final CAD parameter combinations (best combinations) for all designs compared pairwise. Darker shaded boxes mean greater dissimilarity.

Naïve Bayes classification. This shows that with as few as 8–10 features we can already achieve high classification scores of 0.8 AUC and higher. The `ensemble` approach permits us to use only eight features to achieve high scores. Across different benchmarks, `ensemble` provided robust superior performance. We use this approach for subsequent analysis.

We investigate the effects of proper selection of feature counts and training size for the `net_chn4` benchmark in Figure 9. We observe a *Goldilocks* zone of 8–22 features at which the AUC values are maximized. Too few features or too many features result in poor classification quality. The larger balloons also show a clear trend favoring more data for improving prediction quality when using the ideal number of features. Even in presence of limited samples, we are still able to observe high AUC values in the *Goldilocks* zone.

We also seek to understand the simplicity of convergence and combination of feature counts and training sizes that delivers closure. Hence, we perform a sensitivity analysis of feature size and training size to understand when best results are achieved. We need to know if there are certain combinations to avoid from the perspective of eventual convergence. In Figure 10, we plot all possible combinations of features and training sample counts that result in a predictive model with an AUC>0.8 to show the effectiveness of our approach. As the right half (and particularly lower right) of the plot is nearly empty, this suggests training with all available features will not deliver a good predictive model. Instead, if we use a 10–20 features to run the machine learning routines, we see a high density of feasible high-AUC configurations. In particular, even with few training samples, we are able to train high quality models, which is highly relevant for FPGA CAD runs as they are typically slow and training data is expensive to generate.

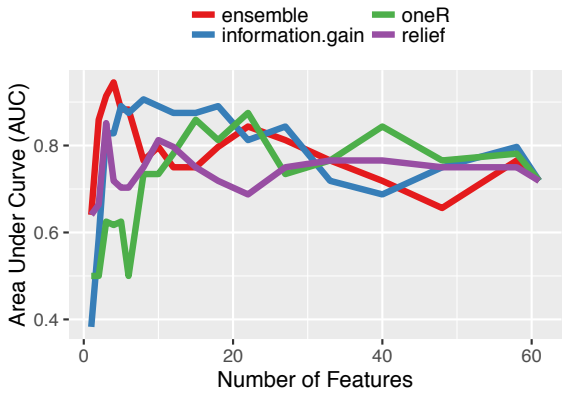


Fig. 8: The Benefit of Feature Selection shown for `medical_jpn` benchmark (trained with *Naïve Bayes*).

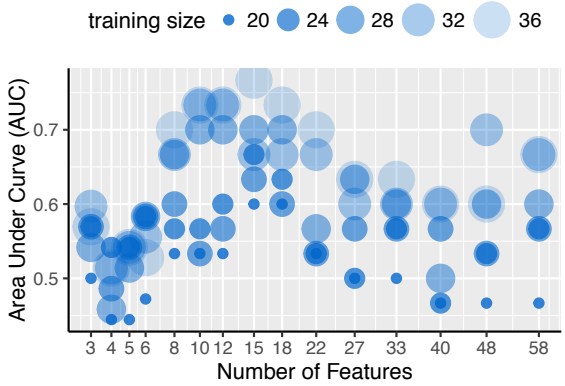


Fig. 9: Impact of Feature Size on AUC trends for varying Training Set sizes for the `net_chn4` benchmark. Balloon sizes correspond to training size. Too few features < 8 result in low AUC scores of 0.5–0.6. Too many features result in a dropoff in AUC scores again. *Goldilocks* range of features 8-22 deliver high AUC scores of 0.7–0.8.

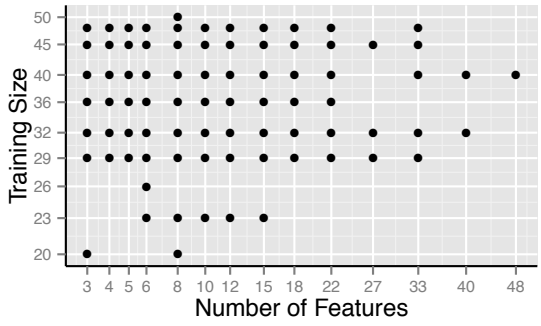
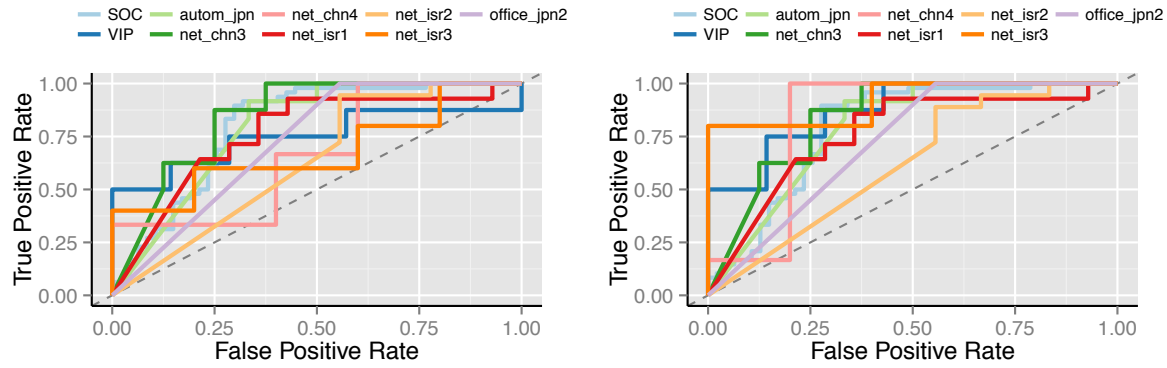


Fig. 10: Showing combinations of Feature Counts and Training Size that deliver AUC>0.8. Black dots in the grid correspond to combinations that result in high quality classifier. Absent dots in the grid mean those combinations delivered poor classifiers.



(a) ROC curve of baseline *Naïve Bayes* model

(b) ROC curve of ensemble model

Fig. 11: Improvement in ROC curves trends with our approach. The original InTime approach delivers ROC curves that hug the diagonal line in some instances. The improved Ensemble classifier leans to the top-left corner of the plot delivering higher predictive quality.

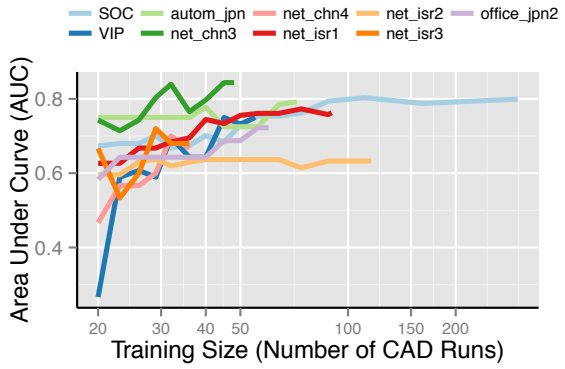


Fig. 12: Impact of Training Samples on AUC scores for *Naïve Bayes* Technique. As expected, more training samples result in better prediction. Most designs converge in 50–100 runs, `SOC` runs for 200+ iterations and does not close timing (see Figure 5).

C. Understanding Effectiveness of Learning

When comparing machine learning algorithms, we are interested in understanding the metrics for these techniques under limited number of training samples as well as a pruned set of attributes (CAD tool parameters). In Fig. 12, we show these trends for the representative *Naïve Bayes* approach. Each circuit needs a minimum set of samples to start effective classification ($\approx$ 20–30 samples). As we increase the number of samples, we observe an improvement in AUC scores. In most cases, it took around 40 samples to build a trustworthy model; and AUC score seemed to converge at around 90 training sample. InTime runs 30 CAD instances in each *round*. We see improvements approximately up to 5-6 rounds (150-180 runs), and beyond those we see saturation or early timing closure (hence a few missing points). To clearly understand the relative benefits of our new classification approach over the baseline InTime technique, we perform a robust design-space exploration across the large industrial designs as shown in Figure 11. Here, we compare the original *Naïve Bayes* with our Ensemble approach. It is clear that the ROC curves of Ensemble models leans towards the top left corner more than those of the baseline models, which translates into that higher predictive quality for Ensemble technique.

D. Unifying Classification and Feature Selection

When we consider the combined effect of features selection and classification together, a somewhat different picture emerges from the experiment. When simply extending original InTime approach with more robust classifiers as shown in Figure 13, the Ensemble approach (`ensem`) deliver higher AUC scores of 0.8 over the original baseline score of 0.74. This is promising as it helps reduce the number of CAD runs required to achieve timing closure. However, when we consider Feature Selection (pruning the set of features to 8–22 features instead of the whole set of $\approx$80), we see that the relative AUC scores of almost all classifiers is now substantially better. Ensemble approach deliver an AUC score of 0.82 while Stack and Bagging deliver a marginally higher score of 0.83. This suggests that design-specific filtering of CAD parameters prior to classifier model training is crucial and has a greater impact on the final prediction quality. Having established the importance of feature selection, it is still worthwhile to note that the exact classifier that delivers best performance for a given design is different for each design.

Finally, in Figure 14, we show the reduction factor in number of CAD iterations when comparing original InTime baseline against our improved approach. Most iterations for a given design take similar amount of runtime so a direct iteration ratio captures the degree of saving in CAD effort.

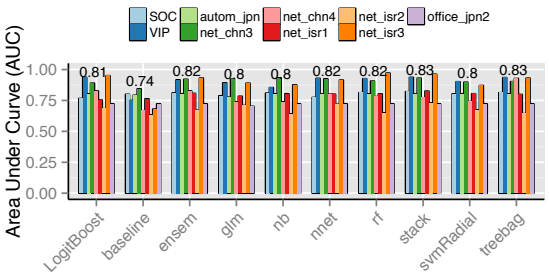


Fig. 13: Combining Feature Selection and Classifier Selection. Largest improvements over baseline are possible when using Feature Selection followed by Classification. The different classifiers shown on the x-axis have been previously explained in [13].

As we see, we are able to deliver a 2–8× reduction in iteration counts thereby speeding up the delivery of timing convergence. The `SOC` design is an exception as it does not meet timing, but quits early resulting in 8× saving. When excluding this design, we notice a 2–6× reduction (mean 3×) in the number of iterations. This shows how exploiting the unique characteristics of the FPGA design as part of the CAD flow can not only improve AUC scores but also translate into wins for acceleration of timing closure.

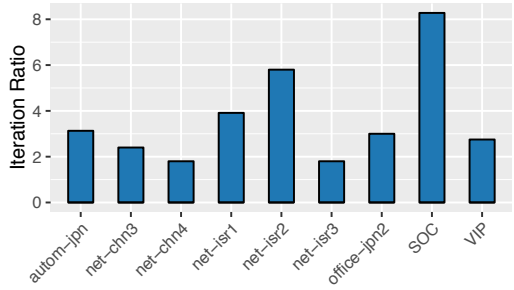


Fig. 14: Iteration Count Reduction when using Feature Selection+Classification vs. original InTime baseline.

V. Discussion

Early on, we observed a preference for design-specific feature selection as a way to help prune the number of features used during the training process. This is critical as each FPGA design has unique characteristics and may be influenced by a varying subset of CAD parameters. We subsequently investigated ways to reduce the time to timing closure and identified ensemble-based methods doing better than other individual approaches. However, the gap between the various classification algorithms was not as significant once we performed feature selection prior to the training phase.

Target	Domain	Gap	Cite
ATLAS	Linear-Algebra	92%	[12]
μarchitecture	Comp. Arch.	10×	[5]
*g*cc Compilers	C/C++	20%	[11]
VPR	FPGA Arch.	17–110%	[8]
Soft-Processors	Comp. Arch.	17×	[10]
HLS LLVM	C/C++	16%	[2]

TABLE III: Related Work

In Table III, we highlight other allied domains where intelligent heuristics have been applied to optimize a mapping and report the extent of optimizations that were uncovered through the use of such intelligent exploration. In [12], a design-space exploration is performed to optimize scientific computing code for specific supercomputing architectures to enable high-performance library implementations on those platforms. [5] shows how to prune the search space of μarchitecture design for complex modern processor pipelines using machine learning heuristics. In [11], the output of `gcc` compilations are optimized by tweaking compiler switches using machine learning to guide the performance tuning. The open-source FPGA development tool `vpr` is amenable to solution optimization by exploring various parameters in the underlying heuristics [8]. A Design of Experiments approach used in [10] for optimizing various soft processor organizations unlocks a large potential for possible improvements and shows how to systematically exploit it. The effect of compiler optimizations on the quality of generated RTL code was explored in the context of the LegUp [2] compiler. Our work targets the rising complexity of the modern FPGA CAD toolchain by using machine learning heuristics to deliver timing closure for different designs.

VI. Conclusions

InTime is a machine-learning based plugin for FPGA CAD tools that automates the delivery of timing closure for FPGA designs. In this paper, we significantly improve the prediction quality of the machine learning flow by combining design-specific feature selection with classification. Each FPGA design exhibits unique timing characteristics that can be exploited to select the most relevant subset of CAD parameters (features) for training a high quality classifier. Using this approach, we are able to reduce the number of required features down to 8–22 features for most designs (75% reduction in model complexity) while boosting the AUC (area under ROC curve) scores for the classifier from 74% to 83%. This directly translates into 3× mean reduction in the number of CAD iterations required to build the classifier model thereby delivering faster timing closure with fewer CAD iterations. As part of future work, we seek to build post-synthesis models that help drive timing convergence using design-style specific recipes in addition to pruning the feature set.

References

[1] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, and L. Scrucca. *caret: Classification and Regression Training*, 2015. R package version 6.0-52.

[2] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, S. Brown, and J. Anderson. The Effect of Compiler Optimizations on High-Level Synthesis for FPGAs. *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pages 89–96, 2013.

[3] N. Kapre, B. Chandrashekaran, H. Ng, and K. Teo. Driving timing convergence of fpga designs through machine learning and cloud computing. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 119–126, May 2015.

[4] N. Kapre, H. Ng, K. Teo, and J. Naude. Intime: A machine learning approach for efficient selection of fpga cad tool parameters. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 23–26, New York, NY, USA, 2015. ACM.

[5] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM Request Permissions, Nov. 2006.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[7] P. Romanski and L. Kotthoff. *FSelector: Selecting attributes*, 2014. R package version 0.20.

[8] R. Y. Rubin and A. M. DeHon. Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-pathfinder. In *FPGA '11: Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM Request Permissions, Feb. 2011.

[9] Y. Saeys, I. Inza2, and P. Larranaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):25072517, 2007.

[10] D. Sheldon, F. Vahid, and S. Lonardi. Soft-core Processor Customization using the Design of Experiments Paradigm. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[11] S. Triantafyllis, M. Vachharajani, N. Vachharajani, and D. I. August. Compiler optimization-space exploration. *Code Generation and Optimization, 2003. CGO 2003. International Symposium on*, pages 204–215, 2003.

[12] R. C. Whaley and J. J. Dongarra. *Automatically tuned linear algebra software*. IEEE Computer Society, Nov. 1998.

[13] Q. Yanghua, N. Kapre, H. Ng, and K. Teo. Improving classification accuracy of a machine learning approach for fpga timing closure. *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, 2015.