# Placement and Routing for 3D-FPGAs using Reinforcement Learning and Support Vector Machines

R. Manimegalai E. Siva Soumya V. Muralidharan B. Ravindran V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras, India

D. Bhatia
Department of Electrical Engineering, University of Texas at Dallas, USA

## Abstract

*The primary advantage of using 3D-FPGA over 2D-FPGA is that the vertical stacking of active layers reduce the Manhattan distance between the components in 3D-FPGA than when placed on 2D-FPGA. This results in a considerable reduction in total interconnect length. Reduced wire length eventually leads to reduction in delay and hence improved performance and speed. Design of an efficient placement and routing algorithm for 3D-FPGA that fully exploits the above mentioned advantage is a problem of deep research and commercial interest. In this paper, an efficient placement and routing algorithm is proposed for 3D-FPGAs which yields better results in terms of total interconnect length and channel-width. The proposed algorithm employs two important techniques, namely, Reinforcement Learning (RL) and Support Vector Machines (SVMs), to perform the placement. The proposed algorithm is implemented and tested on standard benchmark circuits and the results obtained are encouraging. This is one of the very few instances where reinforcement learning is used for solving a problem in the area of VLSI.*

**Keywords**: *Three-Dimensional FPGA, Placement and Routing, Reinforcement Learning (RL), Two-opt algorithm, Support Vector Machines (SVMs).*

## 1. Introduction

The astonishing accuracy with which Moore's law [28] has been adhered to all these years has been the pride of the semiconductor field. Studies have shown that in the near future, there possibly exists a saturation of Moore's law. The only possible solution to this could be a paradigm shift in the fabrication process, which also warrants major changes at the architectural level of VLSI chips. With the advent of 3D-VLSI, Moore's law would still continue to hold good by providing solutions to the challenges of increased speed and decreased power consumption. Three-Dimensional Field Programmable Gate Arrays (3D-FPGA) are one of the most interconnect dominated devices and can benefit the most by 3D integration. The reason being, the average Manhattan distance between any two elements in 3D-FPGA is less when compared to that in 2D-FPGA, due to the vertical stacking of the silicon layers [1, 2, 14, 25]. Some of the fabrication techniques to realize these 3D-VLSI circuits are discussed in [23, 24, 6].

Conceptually, a 3D-FPGA architecture is realized from 2D-FPGA by replicating several layers of 2D-FPGA stacked one over the other and replacing the 2D-switch blocks in every layer by the 3D-switch block. The increase in the number of nearest neighbours increases the number of paths/resources for routing hence leading to a considerable reduction in wire-length and congestion in 3D-FPGA.

**Definition 1** (**Placement and Routing Problem on FPGA**): *Given a synthesized net-list of a circuit which consists of the logical elements and the interconnections between them, compute*

1. *a mapping of the different logical elements in the net-list on to the configurable logic blocks of the FPGA (Placement); and,*
2. *a mapping of the interconnects in the net-list on to the routing resources of the FPGA such that, the connectivity between various logical elements as specified in the net-list is maintained (Routing);*

*such that, the total interconnect length and the channel width (maximum number of interconnects incident on every side/face of a switch block in the FPGA) are minimized.*

The reason for using RL and SVMs for VLSI Placement and Routing is as follows: The Placement and Routing problem on FPGAs is NP-complete and to search for a best solution in the solution-space is very hard. The above is an instance of a combinatorial optimization problem that fits very well into the framework of local search based solution techniques [9]. Reinforcement Learning [19] (RL) has been successfully employed to improve the performance of various local search algorithms [4, 10, 21, 27]. A RL based method for enhancing the local search is presented in [21]. As RL has been proved effective in solving local search based methods, it can also be applied to the VLSI placement and routing problem. Support Vector Machines (SVMs) captures the essential details of the problem instance even with sparse sample data and proved to be effective for high dimensional input space. In this article, we extend the work of [21] and use SVM with RL to enhance the *outcome of two-opt*, a well-known local search heuristic. The salient features of our work are:

1. The proposed solution is one of the very few instances, wherein, RL is used for solving a problem in the area of VLSI.
2. The learning process models the common features among the different instances of the placement and routing problem on FPGA and hence is instance independent. In other words, learning done on one set of representative benchmark circuits is used to place and route the remaining benchmark circuits effectively.
3. This is one of the first applications to use SVM in conjunction with a batch RL algorithm [7].

Next section explores the previous work done in 3D-FPGA and RL. Basic concepts of RL and SVM are given in section 3. Section 4 describes the proposed algorithm, gives the details of two-opt algorithm, features that are used to capture the essential characteristics of a placement, and three phases of the approach adopted. Experimental Results are presented in section 5. Section 6 concludes the paper and gives some open issues to be explored further.

## 2. Related Work

Two-dimensional FPGA architectures are well studied and reported in the literature [12]. There are many algorithms available for Placement and Routing for 2D-FPGA [16, 17, 15, 26]. On the other hand, though roots of 3D-FPGA started way back in 1996, not much has been achieved in practical aspects. Design and development of tools and analysis of 3D circuits complete a major step towards realization and commercialization of this technology [24]. Details of Rothko, a three-dimensional architecture, is discussed in [25]. Universal switch blocks for three-dimensional FPGA design are presented in [8]. Shamik Das et al. have explored the performance and wiring requirement for three-dimensional Integration of Field-Programmable Gate Arrays in [6, 18]. A physical layout of 3D-FPGA and a Spiffy Tool for simultaneous placement and routing is given in [1, 2, 14]. A solution to the channel-routing problem using Reinforcement Learning methods is presented in [10]. To the best of our knowledge, [10] is the only instance reported in the literature, where, RL is used to solve a problem in the area of VLSI.

## 3. Reinforcement Learning and Support Vector Machines

Reinforcement Learning (RL) refers to a collection of learning algorithms that seek to approximate solutions to stochastic sequential decision tasks with scalar evaluative feedback. RL algorithms are designed to operate online and in close interaction with the environment in which the agent is operating. [19] is a good introduction to RL. The formalism usually adopted in RL is that of Markov decision processes, defining the interaction between agent and environment in terms of states, actions, stochastic dynamics, and scalar evaluations or rewards. The goal of an RL agent is to learn a mapping from states to actions, known as a policy, so as to maximize some long term measure of performance.

The concepts of value and value functions are the key features of the reinforcement learning methods. The value of a state under a particular policy is the expected long-term reward an agent will receive, given that the agent starts in that state and follows the given policy. The policy that gives the maximum possible value for each and every state is known as an optimal policy and the corresponding value function is known as the optimal value function. Hence the goal of the agent is to learn an optimal policy.

An RL agent maintains an estimate of the value function that it updates over time. Given some trajectory experienced while following a policy $\pi$, we use a *first-visit Monte Carlo* [19] method to update the value function, $V$. The value function is updated as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)) \tag{1}$$

where, $s_t$ is the state at time t, $\alpha$ is a constant step size parameter, and $R_t$ is the cumulative scalar evaluation, or *return*, received after time $t$, while following $\pi$. This formulation assumes that every trajectory encountered terminates eventually.
Representing this estimate of the value function in case of large problems requires the use of some form of function ap-

proximation mechanism. In this work, we choose to employ regression with *support vector machines* (SVMs). SVMs are a popular tool for solving non-linear classification and regression problems. They work on the principle that if a non-linear classification/regression problem is transformed to a high dimensional space, then it is more likely to be linear in that space. In practice SVMs have been shown to work very well with sparse training data and on very high dimensional input spaces. Since we anticipated that we might not be able to generate too many training samples, we decided to go in for an SVMs based function approximator. For a comprehensive introduction to SVMs see [22].

We use a Gaussian kernel based SVM for the training. The value function for hill climbing is determined as follows:

$$V(\overline{f}) = \sum_{i=1}^{n} \alpha_i \times K(\overline{f}, \overline{f}_i)$$

where, n is the number of support vectors, $\overline{f}$ is the input feature vector, $\overline{f}_i$ is the $i^{th}$ support vector, $\alpha_i$ is the weight corresponds to the $i^{th}$ support vector and K is the Gaussian kernel function given by the following equation.

$$K(\overline{f}, \overline{f}_i) = e^{-\left(\frac{(|\overline{f} - \overline{f}_i|)^2}{\sigma^2}\right)}$$

where, $\sigma$ is the standard deviation whose value can be chosen depending on the accuracy desired in the curve fitting process.

We use *SVM Torch* [20], an open-source implementation, for training the SVMs in our experimentations. The training phase requires us to supply a set of input vectors and corresponding target values. Thus, we generate a set of states and their corresponding returns from each trajectory and train the SVM only after a batch of trajectories are completed. This approach is similar to the first approach described in [7].

## 4. Methodology for Instance Independent Routing

Our approach in this paper is based on earlier work by Moll et al. [21] on applying RL to combinatorial problems. The main idea is to employ RL to improve the performance of local search algorithms. Local search algorithms start with some initial random solution to the problem and progressively improve on this solution by searching in the neighborhood of the current solution for a better solution. The quality of the final solution found depends on the initial solution. Typically the local search algorithm is run multiple times and the best among the different solutions produced is retained. We improve upon the performance of a given local search algorithm by employing RL to learn a good starting point for the local search process, one that leads to a good final solution. Hence, finding a state with high value function would lead to a good final solution.

We employed shortest path routing, since we wanted to use a fixed low cost routing algorithm. Hence the solution space is characterized by the placement alone. The “goodness” of a solution, or placement, is given by the quality of the routing produced by the routing algorithm for this placement. Quality or cost of a solution is measured in terms of the interconnect length and channel width and is given by:

$$cost = \frac{\Big(WireLength + (\alpha \times ChannelWidth)\Big)}{Numberoflinks}$$

where, a link is a connection between two logic blocks in a net. The wire-length is the total interconnect length of the circuit i.e. all the links in all nets put together and $\alpha$ is the channel-width weight. Numerically, the channel-width is negligible compared to the interconnect length. Hence, to give weightage to channel-width the alpha value is used. After some experimentation, we found that setting $\alpha$ to 5 provides a good trade off between wire length and channel width.

### 4.1. Features

A complete description of a placement leads to an intractably large state space. In addition, the value function we learn on one problem will not transfer from one to another. Hence we represent each placement by a set of features, which have been suitably normalized to make them *instance independent*, i.e., the range of values taken by the features is fairly invariant across problems of different sizes and complexity. After some initial experimentation, we have chosen a few features which capture the salient characteristics of the placement and routing for a circuit. Each feature is explained further below. However, it is worth mentioning here that there is an upper limit placed on the channel-width of a switch block for every circuit. This does not curb the placement and routing algorithms. If a switch block has a higher channel-width than the prescribed limit after routing then it is said to be a conflict.

**Description of Features of a Placement:**

1. *LengthFit:*
   The maximum possible interconnect length depends on the size of the FPGA. Given a 3D-FPGA of size, $x \times y \times z$, assuming every switch block reaches its upper limit on channel-width, the maximum total length of all interconnect wires between the switch blocks is

$$\Big((x(y+1) + (x+1)y)z + (x+1)(y+1)(z-1)\Big)W$$

where, W is the upper limit on the channel-width. Length fit is the ratio of the actual total interconnect wire length after routing to the maximum wire length estimated as given above. Thus the lower the actual total wire length, the lower the *LengthFit* value.

2. *Switch Block Congestion:*

   This feature is chosen to maintain uniform load across the channels of the switch blocks in the FPGA. This however, does not directly take the numerical value of the channel width into account but it aims at distributing the channel-width equally through out the FPGA.

   All the switch blocks are sorted in the increasing order of their channel-widths. The channel-widths for the first half of the sorted list of switch blocks is summed up and this forms the lower sum. Similarly, the channel-widths of the second half of the sorted list of the switch blocks is also summed up to give the upper sum. The fractional increase is used as the feature. The lower value of this feature keeps the load distributed throughout the FPGA.

3. *Conflict Ratio:*

   It gives an estimate of the number of switch blocks whose channel-width exceeds the upper limit placed (as defined above). The fraction of switch blocks where the conflict occurs is taken as a feature.

4. *Features corresponding to the link length:*

   Unit link length for a net is computed as the ratio of the link length to the number of links in that net. The maximum and the minimum of the unit link length over the entire circuit, the mean of the first, second and third maxima of the unit link lengths, and the mean of the first, second, and third minima of the unit link lengths are all used as features.

5. *Cube net ratio*

   This feature accounts for the characteristics of the actual placement. First the size of the smallest cube that can enclose a net is calculated for every net. For this, the number of distinct logic blocks contained in the net is to be evaluated. The maximum diameter of the smallest cuboid that can actually enclose the net is calculated. The ratio of this diameter to the size of the smallest cube needed to encompass the net is computed for every net. These ratios are summed up for the first quarter of the nets and this forms the feature.

6. *Maximum of average link length*

   The maximum of the average link length for all nets is yet another feature. Similarly the minimum of the average link length for all nets is also a feature.

## 4.2. Two-Opt

The local search algorithm that we adopt for 3D-FPGA placement is the Two-Opt algorithm [9]. The state of the system is given by the current placement. Swapping a pair of logic blocks transitions the system to a new neighboring state. The set of all the states, resulting from a single swap forms the *two-opt neighborhood* of the original state. We consider all possible swaps between any two logic blocks. The swap which gives the best cost improvement is retained. This process is repeated until there are no further cost improvements. The two-opt algorithm is applied in solving various problems such as traveling salesman problems [9] and the Dial-A-Ride problem [21]. Variants of the two-opt algorithm are also used in placement and routing of 3D-FPGAs [2]. The two-opt algorithm for placement is given below.

```
function Two-Opt
input: Initial random placement
output: Better placement that yields a reduced two-opt cost
    1. Start with an initial random placement.
    2. Search the two-opt neighborhood for the best swap
    3. Perform the swap and move to the next state
    4. Repeat steps 2 and 3 till no further improvement is possible.
endfunction
```

## 4.3. Learning

In order to demonstrate the instance independent nature of our approach, we use only two benchmark circuits during learning, while we test the performance of our approach on a suite of benchmarks from [1]. The benchmarks we chose for learning are *term1* and *2large*. During learning, we operate in two phases. In the first phase, we start with some initial random placement of the given circuit. We consider the two-opt neighborhood of this placement and perform a modified hill-climbing on the estimated *value function* in a first-improvement fashion. In other words, we order the neighbors randomly and pick the first one that yields an improvement of at least $\epsilon$ in value over the current placement. The current placement then receives an evaluation of $-\epsilon$. This small penalty is to ensure that the search trajectories do not become too long.
If no such neighbors leads to an improvement, we terminate the trajectory and run two-opt starting from the final solution. The evaluation for this final solution is the negative of the cost of the two-opt solution. Thus for any placement on a trajectory of length $n$, the return is given by:

$$-(n-i-1)\epsilon - costOfTwoOpt$$

We store the features and the return corresponding to each placement seen along the trajectory. After we accumulate several such trajectories we go to the second phase, where we use all the data collected so far to train an SVM to improve our value function estimate. An input vector to the SVMs is given by the features corresponding to a placement and the target is the corresponding return.

If necessary we can repeat phases 1 and 2 multiple times to improve the value function estimate. In each repetition we retain the data accumulated in the previous executions of phase 1. In our experiments we used 50 trajectories for each benchmark and repeated each phase twice. It should be mentioned that the memory required during initialization and training the SVMs is not high (478k), and the output generated by the SVM tool stores the support vectors requires very less space (1.2Kb). Training the SVM multiple times does not need much time.

## 4.4. Initialization

Since we used a batch mode to train our SVMs, we need to initialize it to some reasonable value estimates before starting the learning process. In order to do this, we decided to use the cost of the two-opt solution generated by starting from some random initial placement. Thus the SVM was trained with input vectors describing initial random placements and targets given by the corresponding two-opt solution costs. We used 100 initial placements for each benchmark. When we performed the SVM training in phase 2 for the first time, we retained these initial placements in the training set, while for the subsequent iterations we ignored these.

## 4.5. Application

While employing the learned value function, we just perform phase 1 of the learning procedure. We start with some initial random placement of the circuit, then hill climb on the estimated value function in a first improvement fashion using the two-opt neighborhood. When the trajectory terminates, we run two-opt from the final placement. The resulting placement is taken as the output of the algorithm. If necessary this process maybe repeated for multiple starting points and the best solution obtained is retained.

## 5. Experimental Results

The above proposed algorithm is implemented in C++ and the results obtained are given in this section. We have tested our proposed algorithm for four layers. In tables 1 and 2,

| Circuit | InterconnectLength | | | ChannelWidth | | |
|---|---|---|---|---|---|---|
| | Mo. | RL | $\Delta$ | Mo. | RL | $\Delta$ |
| 9symm1 | 583 | 428 | 26.58 | 6 | 5 | 16.67 |
| 2large | 1412 | 1101 | 22.02 | 6 | 6 | 0 |
| alu2 | 1282 | 1089 | 15.05 | 6 | 6 | 0 |
| apex7 | 713 | 500 | 29.87 | 7 | 4 | 42.85 |
| example2 | 1265 | 857 | 32.25 | 6 | 5 | 16.67 |
| term1 | 452 | 251 | 44.46 | 6 | 3 | 50.00 |

**Table 1. Comparison of Mondrian algorithm with RL based placement and routing**

| Circuit | Minimum cost | | Average cost | | Cost |
|---|---|---|---|---|---|
| | 2-opt | RL | 2-opt | RL | $\Delta$ |
| 9symm | 1.352 | 1.340 | 1.585 | 1.562 | 0.87 |
| 2large | 1.675 | 1.604 | 1.923 | 1.867 | 4.23 |
| apex7 | 1.232 | 1.159 | 1.495 | 1.506 | 5.90 |
| term1 | 1.041 | 0.917 | 1.243 | 1.196 | 11.92 |
| example2 | 1.359 | 1.313 | 1.602 | 1.616 | 3.32 |
| alu2 | 1.696 | 1.687 | 1.976 | 1.990 | 0.45 |

**Table 2. Comparison of cost measure**

Mo. denotes the Mondrian algorithm and $\Delta$ denotes percentage improvement obtained using the proposed approach over the Mondrian approach. Table 1 compares the results with the algorithm proposed in [2]. From Table 1 it can be observed that the wire-length is reduced for all the circuits when compared to the Mondrian approach. The channel width remains same or reduced for all the circuits. This clearly demonstrates the superiority of the proposed RL based approach for the placement and routing problem on 3D-FPGA over the existing algorithms. Table 2 compares the cost of the same initial placement before applying RL (only Two-opt) and after applying RL for different benchmark circuits.

## 6. Conclusion

In this paper we propose Reinforcement Learning based approach using Support Vector Machines for the Placement and Routing problem on 3D-FPGA. Our experimental results on standard benchmark circuits indicate that our approach produces effective routing, with less interconnect length and channel width. Even with a few training trajectories, RL combined with SVMs gives a surprisingly good performance. This indicates that by providing more training examples our method can achieve better performance. The approach presented in this paper is instance independent by virtue of the chosen set of features. Further effort can be directed towards the design of better feature set leading to greater improvement in performance. In this work, fixed routing algorithm is assumed. We can also use RL techniques to improve on the existing routing algorithm. Another interesting direction of research is to do power analysis and incorporate thermal and fabrication issues in our cost function leading to power aware placement and routing.

## References

[1] M. Alexander, J. P. Cohoon, J. L. Colflesh, J. E. Karro, and G. Robins, "Three-Dimensional Field-Programmable Gate Arrays", *In Proceedings of the IEEE International ASIC Conference*, Austin, TX, September 1995, pp. 253-256.

[2] M. Alexander, J.P. Cohoon, J. Karro, E.L. Peters, and G. Robins, "Placement and Routing for Three-Dimensional FPGAs", *In Fourth Canadian Workshop on Field-Programmable Devices*, Toronto, Canada, May 1996, pp. 11-18.

[3] V. Betz, J. Rose, and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, February 1999.

[4] J. A. Boyan, and A. W. Moore, "Using Prediction to improve Combinatorial Optimization Search", , *In Proceedings of AI-STATS-1997.*

[5] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, "Field-Programmable Gate Arrays", *Kluwer Academic Publishers*, Boston, MA, 1992.

[6] S. Das, A. Chandrakasan, and R. Reif, "Three-Dimensional Integrated Circuits: Performance, Design Methodology, and CAD Tools", *In Proceedings of ISVLSI-2003*, Tampa, Florida, February 2003, pp.13

[7] T.G. Dietterich, and X. Wang, "Batch Value Function Approximation via Support Vectors", *In Proceedings of Advances in Neural Information Processing Systems 14*, Vancouver, British Columbia, Canada,, 2001, pp. 1491-1498

[8] Guang-Ming Wu, Michael Shyu, and Yao-Wen Chang, "Universal switch blocks for three-dimensional FPGA design", *International Symposium on Field Programmable Gate Arrays Proceedings of the 1999 ACM/SIGDA*, California, United States, 1999, pp. 254.

[9] H. P. Christos, and Kenneth Steiglitz , "Combinatorial Optimization: Algorithms and Complexity", *Prentice Hall Inc.*, New Jersey, USA, 1982.

[10] J. A. Boyan, and A. W. Moore , "Learning Evaluation Functions for Global Optimization and Boolean Satisfiability", *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998, pp. 3-10.

[11] G. Kelly, A. Morrissey, J. Alderman, and H. Camon, "3-D Packaging Methodologies for Microsystems", *In IEEE Transactions on Advanced Packaging*, Vol.23, No.4, November 2000, pp.623-630.

[12] H. Schmit, and V. Chandra, "FPGA Switch Block Layout and Evaluation", *in Proceedings of Field Progrmmable Gate Arrays-2002* February 2002, California, USA, pp.24-26.

[13] J. H. Holland, "Adaptation in Natural and Artificial Systems", MIT Press, MA, USA, 1992.

[14] J. Karro and J. Cohoon, "A Spiffy Tool for the Simultaneous Placement and Global Routing for Three-Dimensional Field Programmable Gate Arrays", *In 9th Great Lakes Symposium on VLSI* Ann Arbor, Michigan, March 1999, pp. 226-227.

[15] G. G. Lemieux and S. D. Brown, "A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays", *In Proceedings of ACM/SIGDA Physical Design Workshop*, Lake Arrowhead, CA, April 1993, pp. 215-226.

[16] A. Muthukaruppan, S. Suresh, and V. Kamakoti, "A Novel Parallel Three Phase Genetic Approach to Routing for Field Programmable Gate Arrays", *In Proceedings of IEEE Field Programmable Technology 2003*, Hong Kong, pp.336-339.

[17] Parivallal Kannan and Dinesh Bhatia, "Tightly Integrated Placement and Routing for FPGAs", *In the proceedings of International Conference on Field Programmable Logic(FPL) 2001*, pp.639-646.

[18] A. Rahman, S. Das, A. Chandrakasan, and R. Reif, "Wiring Requirement and Three-Dimensional Integration Technology for Field Programmable Gate Arrays", *In IEEE Transactions on VLSI Systems*, vol.11, No.1, February 2003, pp.44-54.

[19] R. S. Sutton, and A. G. Barto, "Reinforcement Learning:An Introduction", *MIT Press*, Cambridge, MA, 1998.

[20] R. Collobert, and S. Bengio, "SVMTorch: Support Vector Machines for Large-Scale Regression Problems", *In The Journal of Machine Learning Research*, MIT Press, MA, USA, Vol 1, September 2001, pp. 143-160.

[21] R. Moll, A. G. Barto, T. J. Perkins, and R. S. Sutton, "Learning Instance-Independent Value Functions to Enhance Local Search", *Advances in Neural Information Processing Systems Conference*, Denver, Colorado, The MIT Press 1999, pp.1017-1023.

[22] Simon Haykin, "Neural Networks, A comprehensive foundation", 2nd ed., Prentice Hall, 1999, 842 pages

[23] S. W. Bond, et. all, "A Three-Layer 3D Silicon System Using Through-Si Vertical Optical Interconnections and Si CMOS Hybrid Building Blocks", *In IEEE Journal of selected topics in Quantum Electronics*, Vol 5, No.2, April 1999, pp.276-285.

[24] S. M. Alam, D. E. Troxel and C. V. Thompson, "A Comprehensive Layout Methodology and Layout Specific Circuit Analyses for Three-Dimensional Integrated Circuits", *ISQED 2002*, pp.246-251.

[25] W. M. Meleis, M. Leeser, P. Zavracky, M. M. Vai, "Rothko:Architectural Design of a Three Dimensional FPGA", *IEEE Proceedings of 17th Conference on Advanced Research in VLSI (ARVLSI '97)*, September, 1997, Ann Arbor, MI, pp. 16-23.

[26] Y. L. Wu, and M. Marek-Sadowska, "An Efficient Router for 2-D Field Programmable Gate Arrays", *In European Design and Test Conference*, 1994, pp.412-416.

[27] W. Zhang, and T. G. Dietterich, "A Reinforcement algorithm learning approach to Job-shop Scheduling", *In the proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995, San Francisco, pp.1114-1120.

[28] http://public.itrs.net/