Improving FPGA Placement with Dynamically Adaptive Stochastic Tunneling

Mingjie Lin, *Member, IEEE,* and John Wawrzynek, *Member, IEEE*

***Abstract*—This paper develops a *dynamically adaptive stochastic tunneling* (DAST) algorithm to avoid the "freezing" problem commonly found when using simulated annealing for circuit placement on field-programmable gate arrays (FPGAs). The main objective is to reduce the placement runtime and improve the quality of final placement. We achieve this by allowing the DAST placer to tunnel energetically inaccessible regions of the potential solution space, adjusting the stochastic tunneling schedule adaptively by performing detrended fluctuation analysis, and selecting move types dynamically by a multi-modal scheme based on Gibbs sampling. A prototype annealing-based placer, called DAST, was developed as part of this paper. It targets the same computer-aided design flow as the standard versatile placement and routing (VPR) but replaces its original annealer with the DAST algorithm. Our experimental results using the benchmark suite and FPGA architecture file which comes with the Toronto VPR5 software package have shown a 18.3% reduction in runtime and a 7.2% improvement in critical-path delay over that of conventional VPR.**

***Index Terms*—Field-programmable gate array (FPGA), placement, simulated annealing, stochastic tunneling.**

I. Introduction

DESPITE advantages such as fast time-to-market, low non-recurring engineering costs, and exceptional fine-grained parallel performance, field-programmable gate arrays (FPGAs) significantly lag behind application-specific integrated circuits in power efficiency and circuit speed [1], [2] due to their high programming overhead. Consequently, FPGAs cannot be widely used in applications that demand high power efficiency and low critical-path delay. Among the key software factors affecting FPGA's overall performance, placement may be the most influential, as it directly determines the relative locations of target circuit blocks and therefore directly affects the final performance and power consumption of routed design. Unfortunately, finding optimal placement/routing in FPGAs is computationally infeasible due to its NP-completeness, hence can only be approximated by heuristic methods such as simulated annealing. However, conventional simulated annealing-based placement often suffer from the "freezing" problem that traps the optimized solution in local minima, thus degrading the quality of results and prolonging total runtime [3], [4]. For example, experiments [5] have shown that for some large benchmarks with known optimal placement, state-of-the-art placers such as `PAR` from Xilinx, San Jose, CA, Quartus II from Altera, San Jose, CA, versatile placement and routing (`VPR`) [6] and `PATH` [7] from academia can yield results far from the optimal solution. In the extreme case, where each circuit consists of global connections only (G-PEKU benchmarks), these tools can be as far away from the optimal by 41% to 102%. As integrated circuit (IC) technology continues to scale and move toward 20 nm, the FPGA placement problem will only get worse as the number of look-up tables (LUTs) per FPGA continues to increase [2].

Manuscript received July 28, 2009; revised January 11, 2010 and May 10, 2010; accepted June 2, 2010. Date of current version November 19, 2010. This work was supported in part by DARPA, under Contract FA8650-09-C-7907, by the NIH, under Grant 1R01CA130826-01, and by the Berkeley Wireless Research Center (BWRC). This paper was recommended by Associate Editor K. Bazargan.

M. Lin is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA (e-mail: mingjie.lin@gmail.com).

J. Wawrzynek is with the University of California, Berkeley, CA 94720 USA, and also with the Lawrence Berkeley National Laboratory, NERSC Division, Berkeley, CA 94720 USA (e-mail: johnw@eecs.berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2061670

A. FPGA Placement Problem

Conceptually, placement in FPGAs is a graph matching problem. As illustrated in Fig. 1(b), routing resources in a FPGA, such as switch blocks, connection blocks, and interconnect channels can be abstracted as a graph $G_{\text{FPGA}} = (U, E(U))$, comprising a set of vertices U and a set of edges $E(U)$. Given a vertex $u \in U$, we indicate with $E(u)$ the set of neighboring vertices of u (or neighbors, for short), which represents the possible reconfigurable interconnections in the FPGA. Likewise, as shown in Fig. 1(c), a synthesized user design after technology mapping can also be represented as a graph $G_{\text{design}} = (V, E)$ consisting of a set of vertices V and a set of edges $E(V)$. Given graphs G_{FPGA} and G_{design}, the objective of placement is to find an optimized graph mapping so that the performance of the mapped design is maximized according to a pre-defined cost function $C(G_{\text{FPGA}}, G_{\text{design}}, M)$, where M is a mapping function between G_{FPGA} and G_{design} subject to various placement constraints. The objective of FPGA placement is not only to minimize hardware resource usage but also to produce a compact placement that facilitates routing and achieves high operating frequency.

Algorithmically, placement in FPGAs is a NP-hard global optimization (GO) problem, which often faces a solution space so enormous that a deterministic and efficient solution is infeasible. Fortunately, for most GO problems such as predicting protein structures by minimizing its free energy [8], near-optimal solutions can be achieved by probabilistic methods

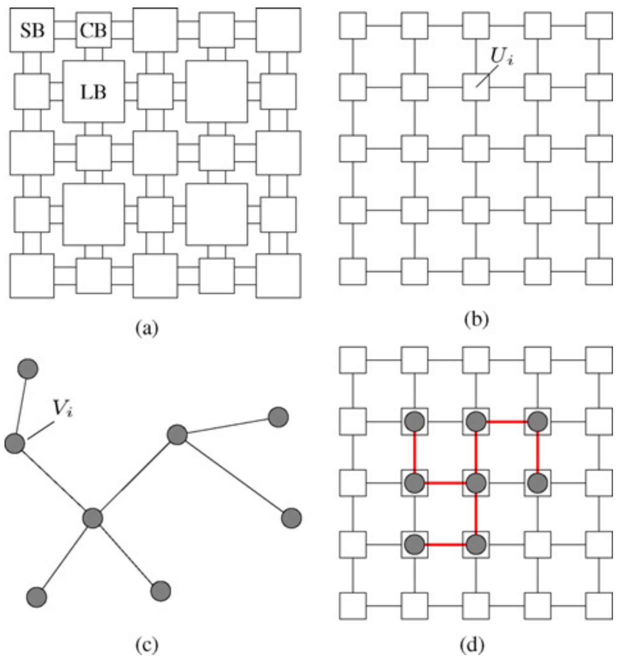


Fig. 1. (a) Baseline FPGA architecture. (b) Graph representation of routing structure. (c) Graph representation of user design circuit. (d) Simple example of a placed and routed design or mapped graph. LB: logic block, SB: switch block, CB: connection block, U_i: routing graph node, V_i: circuit graph node.

with reasonable performance and acceptable runtime. One such heuristic is Monte-Carlo-based sampling, whereby a substantially smaller subset of the otherwise vast solution space is sampled with the Metropolis criterion [9] followed by identifying the optimal solution in this reduced subset. In FPGA literature, such an optimization technique is typically known as the simulated annealing (SA) algorithm, and remains widely used for FPGA placement today as it can readily adapt to realistic architectural constraints.

B. Prior Work

Due to its importance, there have been many attempts to improve the quality and runtime of FPGA placement. The majority of these studies focus on specific heuristics closely related to particular FPGA architectures (e.g., [10]–[12]). This paper instead focuses on improving the simulated annealing algorithm commonly used to drive FPGA placement, which is orthogonal to other FPGA-specific approaches and can potentially be applied to other problem domains. Coarsely, prior work on this subject can be classified as either software or hardware-based.

Software-based approaches typically target improving various components of a conventional simulated annealing algorithm, such as the initial placement, the annealing schedule, the objective function, or the acceptance criteria. For example, VPR [6] introduced several key improvements to FPGA placement, including the concept of path-based weighting for timing-driven optimization, timing-driven clustering, fast incremental bounding box computation, and an experimentally tuned annealing schedule. Recently, researchers [13] have attempted to achieve better initial placements by coupling annealing with other deterministic placement strategies in hope of reducing the overall annealing time. For example, in [14], recursive min-cut partitioning, a faster placement heuristic, is employed to quickly produce better initial placements, thereby requiring the annealer to run only in a lower temperature regime. In [15], an adaptive strategy was proposed to dynamically alter the annealing schedule to better suit the dynamics of the evoluting system states. Given the importance of the objective function in optimization, several more accurate ones have also been suggested such as (e.g., path-based timing weights, or the incorporation of congestion metrics).

Motivated by the extensive design reuse with intellectual property blocks in FPGAs, several authors have recently focused on more application-specific approaches to improve the placement results. For example, [16] presented both incremental slack/criticality update technique and a reformulated cost function to enhance timing-driven FPGA placement for pipelined netlists. In [10], the authors discussed various intelligent strategies for selecting and placing cells that are interspersed with traditional random moves during annealing, allowing the annealer to converge more quickly and to attain better quality with less statistical variability. While almost all these studies are FPGA-specific in order to achieve high-performance placement solutions, we believe that these problem-specific approaches are completely complimentary with our approach to many FPGA applications.

More recently, partially due to the fast advances in VLSI technology, researchers started to investigate using hardware to accelerate the simulated annealing. In [17]–[19], the authors proposed how a systolic structure can accelerate placement by assigning one processing element to each possible location for a FPGA LUT from a design netlist. They demonstrated that their technique approaches the same quality point as traditional simulated annealing measured by a simple linear wirelength metric, while achieving three orders of magnitude reduction in the total computer-aided design (CAD) runtime. Despite their impressive performance gains, hardware-assisted placement often involves HDL programming, therefore seriously limiting its applicability in many circumstances.

Our proposed dynamically adaptive stochastic tunneling (DAST) approach is software-based, which is often much preferred in real-world applications for compatibility reasons. However, unlike many previously proposed software approaches, the DAST algorithm is not FPGA-specific. Therefore, it not only has wide applicability but also can benefit from other techniques specific to FPGA placement.

C. Contributions

This paper has two objectives: 1) investigating the effectiveness of using stochastic tunneling (STUN) to improve runtime and quality of results (QoR) of FPGA placements, and 2) proposing a DAST algorithm that outperforms both conventional simulated annealing and STUN. To this end, we show that the proposed DAST method not only can be readily incorporated into existing FPGA CAD software, but also significantly improves the overall placement QoR. The main contributions of this paper are as follows.

1) *Stochastic tunneling for FPGA placement*: recent advances in stochastic methods, such as stochastic tunneling

and parallel tempering, have been successfully applied to a wide range of optimization problems, e.g., molecular and DNA folding [20]–[22]. However, such means are practically unexplored in the field of CAD. This paper, to the best of our knowledge, is the first study on the effectiveness of stochastic tunneling in placing FPGAs.

2) *Multi-modal move selection based on Gibbs sampling*: instead of being limited to a single move type, our annealing approach chooses among several move types according to Gibbs sampling. This adaptation strategy can further improve the placement quality and achieve better runtime.
3) *Detection scheme for local minima entrapments*: previous studies [23] and our results have shown that stochastic tunneling, in its originally proposed form, cannot satisfactorily prevent the freezing problem, which motivates us to detect local minima entrapment with the detrended fluctuation analysis (DFA).
4) *Dynamic adaptation of tunneling schedule*: we developed a dynamical annealing schedule, which alternates between conventional simulated annealing and self-adaptive stochastic tunneling whenever local minima entrapment is detected. Furthermore, we propose an effective method to automatically select the key stochastic tunneling parameter γ.

The rest of this paper progresses as follows. Section II introduces simulated annealing and uncovers the key weaknesses of its original form when applied to a global optimization problem. We then show in Section III that applying conventional static stochastic annealing directly to the benchmark problems can only improve the quality of results to a very limited extent while the overall annealing procedure still suffers severely from entrapment in local minima. These limitations motivate us to propose the dynamically adaptive stochastic annealing approach in Section IV. Finally in Section V, we apply the DAST algorithm to standard FPGA placement problems and illustrate the superiority of the DAST algorithm.

II. Simulated Annealing

A well-known search-based heuristic, simulated annealing [24], attempts to find the minimum solution R^* of an objective function $\mathbf{F}$ that takes real values over a set of states $\mathbf{S}$ by numerically mimicking an annealing process, in which a thermal system initially melts at high temperature and then cools slowly until it reaches a stable state (ground state) with the lowest energy [25]. Because the ground state of any physical system at temperature zero will concentrate in the vicinity of the global minimum, simulated annealing will ultimately converge to an optimal solution given a sufficiently slow cooling schedule and suitable cost function. Simulated annealing is effective because it can escape from most local optima by allowing the deterioration in the objective function value controlled probabilistically through the annealing temperature (i.e., probabilistically "climbing" up the hill of potential solution space), which fundamentally deviates from all greedy heuristics.

Today, the simulated annealing method is probably the most prominent example of applying the Metropolis sampling method to GO. It introduces two tricks. The first is the so-called "Metropolis algorithm" [9], in which some "bad" moves that yield higher energy value are still accepted, and therefore allows the solver to "explore" more solution space. Such "bad" moves are allowed using the criterion

$$e^{-\frac{\Delta f}{kT}} \geq \mathbb{R}(0,1) \tag{1}$$

where Δf is the change of cost due to a random move, f is called a "cost function," T is a "synthetic temperature," k is Boltzmann's constant, and $\mathbb{R}(0,1)$ is a random number in the interval (0, 1). The second trick is to strategically lower the "temperature" T, i.e., annealing schedule.

SA algorithm has been widely used for FPGA placement because it can be readily adapted to realistic architectural constraints. A conventional simulated annealing-based FPGA placer typically uses a cost function aiming for best possible logic density or timing, sequentially swaps random cell locations, computes the difference in the cost function of Δf, then determines the acceptance probability of each trial swap with $\min(1; e^{-\frac{\Delta f}{kT}})$ as in (1) (Metropolis criterion) [9]. In other words, during FPGA placement, greedy moves are always accepted, while non-greedy ones are accepted with a probability that exponentially decreases according to the current "temperature" and the difference in the overall cost function. Although quite effective for optimizing placement in FPGAs, simulated annealing, as any optimization approaches based on Monte-Carlo sampling, has notable weaknesses.

1) *Sensitivity to parameters*: studies [26] have shown that choices of annealing parameters can significantly affect the effectiveness of simulated annealing. For example, the initial annealing temperature T_0 not only affects the length of the annealing procedure needed to reach the thermal equilibrium, but also the quality of final results. To confirm this, we placed and routed all 20 largest MCNC benchmark designs [27] using VPR. Our experimental results have shown that, as the starting annealing temperature changes from 10 to 100, the deviation of the resulting critical-path delay can vary up to about 10%. Similarly, the annealing schedule, which determines at what point in the procedure and by how much the temperature T is to be reduced, also plays a significant role in the final performance of simulated annealing. While keeping the starting annealing temperature constant at 10, we changed the linear factor α from 0.80 to 0.99; the resulting critical-path delay deviated by up to 13%. Intuitively, the placer prefers high initial T_0 and a slow cooling schedule, however it is not clear which annealing schedule is optimal. In practice, users are often forced to perform manual tuning, which not only is cumbersome but also will not guarantee optimal performance and results. In addition to the annealing schedule, the move generator is particularly important to the annealing performance. For a given problem, there are often different types of moves that can be taken. For FPGA placement, the move can be a random

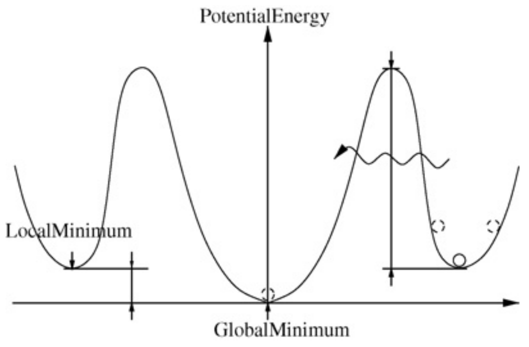


Fig. 2. Schematic 1-D solution space, both local and global minima are illustrated. The current solution is trapped inside a local minimum and can not escape due to the high well surrounding it.

swap, a neighboring swap, or even a random move. Unfortunately, it is unclear as to which move type at any particular time is the optimal choice.

2) *Freezing problem*: many simulated annealing problems with a rugged solution space suffer from the so-called freezing problem [28], occurring when the cooling process ends up in a local minimum (a well of barriers) that cannot be escaped. As depicted in Fig. 2, the current solution reaches one of the local minima, however the energy barrier for the local minimum to escape is far larger than the potential energy difference between the current minimum and the global minimum. Consequently, at high temperatures, the solution can still cross the barriers, but not differentiate between the wells. As the temperature drops, the solution will eventually become trapped with almost equal probability in any of the wells of local minima, failing to resolve the energy difference between them. The success of simulated annealing, as in many other stochastic optimization techniques, hinges on its capability to effectively sample its vast solution space. Unfortunately, when the energy difference between "adjacent" local minima on the topology of solution space is much smaller than the energy of intervening transition states separating them, the freezing problem often arises.

In what follows, conventional static stochastic tunneling is first used to solve the FPGA placement problem. We show that the static stochastic tunneling can only improve the solution to a limited extent, which motivates us to develop two powerful augmentations to static stochastic tunneling, namely, multi-modal moves through Gibbs sampling and dynamically adaptive stochastic tunneling DAST in Section IV.

III. Static Stochastic Tunneling

To improve the performance of simulated annealing, STUN was introduced by Hamacher and Wenzel [29] in the study of physical interacting particles with global order. Its original objective was to overcome large energy barriers while avoiding the "freezing" problem during numerical simulations. The physical idea behind the stochastic tunneling method (STUN) is to allow the particle to "tunnel" forbidden regions of the potential energy surface that are irrelevant for the low-energy properties of the problem.

Static stochastic tunneling can be accomplished by applying the following nonlinear transformation to the solution space (or the potential energy landscape) before applying conventional simulated annealing as in (1)

$$f_{\text{STUN}}(\vec{x}) = 1.0 - e^{-\frac{\mathbb{E}(\vec{x})-\mathbb{E}(\vec{x_0})}{\gamma}} \tag{2}$$

where $\mathbb{E}(\vec{x})$ denotes the value of potential energy at one specific point in the energy space $\vec{x}$, $\vec{x_0}$ is the lowest minimum encountered by the dynamical process thus far [29], and γ is the tunneling parameter that will be discussed in Section IV-D. Location $\vec{x}$ is vectorized because in general, each point in solution space is multi-dimensional. For example, in FPGA placement, $\vec{x}$ denotes one specific FPGA placement. By continuously adjusting the reference energy $f_0 = \mathbb{E}(\vec{x_0})$ to the best energy found so far, (2) preserves both the effective potential and the locations of all minima, but maps the entire energy space from f_0 to the maximum of the potential onto the interval (0, 1]. Functionally, the transformation by (2) is equivalent to amplifying the topology of the local potential landscape and eliminating its irrelevant features so that the probability of escaping local minima is enhanced. When applying STUN, the meaning of $\mathbb{E}(\vec{x})$ and $\vec{x}$ depend on specific applications. For example, in this paper of FPGA placement, $\mathbb{E}(\vec{x})$ represents the cost value computed with specific placement cost function for a specific FPGA placement $\vec{x}$.

Alternatively, the dynamic process in stochastic tunneling with fixed temperature T can be interpreted as an equivalent one with a self-adjusting cooling schedule on the original solution space, where the annealing temperature T_{ST} can be approximated as $\exp\left(-\frac{\mathbb{E}(\vec{x_{1,2}})-\mathbb{E}(\vec{x_0})}{\gamma}\right) \cdot T$. In this process, the equivalent temperature rises rapidly when the local energy is larger than f_0 and the particle diffuses (or tunnels) freely through potential barriers of arbitrary height. As better and better minima are found, ever larger portions of the high-energy part of the potential energy surface are flattened out. To illustrate, Fig. 3(a) shows a 1-D example with three local minimum solutions and two global minimum. After minimum f_1 is found, because the annealing temperature is too low, it becomes difficult for the optimized solution to escape from this local minimum and locate the nearby better solutions f_2 and f_3. However, if we can somehow flatten the solution space in all regions that lie significantly above the best estimate f_1 and simultaneously enhance the potential surface landscape around f_2 and f_3, finding globally optimized solution f_3 is much more probable, as depicted in Fig. 3(b) and (c).

There are several issues associated with using static stochastic tunneling directly. First, as shown in [20] and [30], the performance of static stochastic tunneling depends heavily on the form of transform functions, which was corroborated by our paper. Among tunneling function candidates $1-e^{-\gamma(\mathbb{E}-\mathbb{E}_0)}$, $\frac{1-\text{sgn}(\mathbb{E}-\mathbb{E}_0)}{2}\mathbb{E}$, $\tanh(-\gamma(\mathbb{E}-\mathbb{E}_0))$, and $\sinh(-\gamma(\mathbb{E}-\mathbb{E}_0))$, our experiments have shown that the final optimization results can differ by up to 20%. In our paper, $1-e^{-\gamma(\mathbb{E}-\mathbb{E}_0)}$ yields the best results. Second, the performance of static stochastic tunneling

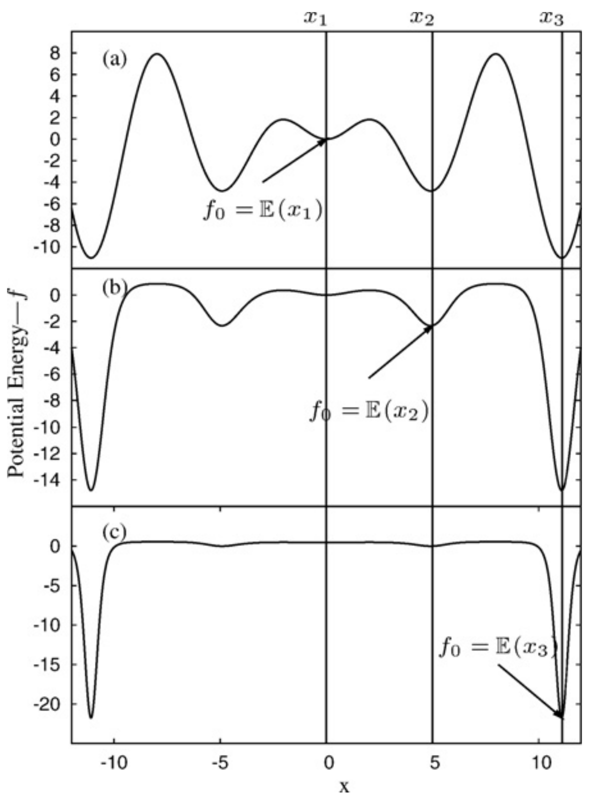


Fig. 3. Schematic of a 1-D solution space, where the minimum indicated by the arrows is the best minimum found so far (f_0). x_1, x_2, and x_3 denote the solution points corresponding to each f_0. Potential energy depicted in (b) and (c) are transformed versions of that in (a) and (b), respectively. The transforming function is $f = \mathbb{E}(x)\left(1.0 - e^{-\frac{\mathbb{E}(x)-f_0}{\gamma}}\right)$. All wells that lie above the best minimum found are suppressed. If the dynamic process can escape the well around the current minimum estimate it will not be trapped by other local minima that are higher.

is sensitive to the choice of the tunneling parameter γ that controls the degree of the cutoff steepness in high-energy regions. As shown in [23], by choosing γ values between 1.0 and 5.0, the final optimization results can differ by up to about 30%, which makes using stochastic tunneling highly unreliable. Consequently, in practice, γ is often manually tuned for optimal performance. Most importantly, our results on FPGA placement have shown that although the QoR from stochastic tunneling is better than that from mere simulated annealing, it is still far from the optimal. All these issues motivate us to develop a dynamically adaptive stochastic tunneling DAST.

IV. Dynamically Adaptative Stochastic Tunneling

Stochastic optimization often levels off in the best minimum found so far, which makes subsequent improvement exponentially difficult. One conceptually appealing approach to circumventing such degradation is to use an adaptive strategy, whereby the effectiveness of optimization is measured on-line and the algorithm is steered according to certain dynamic measures. Such adaptation is not a new idea and has been extensively studied in the context of global optimization. For example, in the energy landscape paving method [31], low-temperature Monte-Carlo simulations were performed with a modified energy expression designed to steer the search away from regions that have already been explored. In [32], a global optimization based on machine learning relied on an internal model of the objective function to represent its state of knowledge. A simple inference network was then used to turn its state knowledge into strategic decisions at each stage of optimization.

Unfortunately, in general applications, no special knowledge about the underlying problem and its respective solution space is available and thus the specific adaptation approaches such as [31] and [32] are not always applicable. We instead develop a dynamically adaptive annealing strategy that requires no manual intervention during its execution while achieving good performance for the FPGA placement problem. In the following, we detail each key component of our proposed DAST algorithm.

A. Multi-Modal Moves Through Gibbs Sampling

When generating candidate moves in simulated annealing, it is essential that after a few iterations of the algorithm, to ensure optimization efficiency, the current state should have much lower energy than a random state. Therefore, as a general rule, one should skew the move generating process toward candidate moves where the energy of the destination state is likely to be similar to that of the current state. This heuristic, the main principle of the Metropolis-Hastings algorithm, tends to exclude "very good" candidate moves as well as "very bad" ones; however, the latter are usually much more common than the former, so the heuristic is generally quite effective.

Typically in problems of simulated annealing, there are different kinds of candidate moves. For example, the traveling salesman problem has at least three kinds of moves: swapping two consecutive cities, swapping two randomly chosen cities, and reversely swapping sections of tour. Similarly for FPGA placement, there are global moves that swap two randomly chosen logic blocks, local moves that move logic blocks within its vicinity, or moves that fine-tune locations of sub-components within a logic block. Intuitively, at the early stage of annealing, global moves tend to be more effective while local moves become more efficient as annealing approaches its end. In short, depending on the temperature and the current landscape of solution space, certain move types tend to be noticeably more effective than others during the process of annealing. The challenge, of course, is to select the most effective move types without prior knowledge of the global solution and the overall solution space. To our knowledge, most annealers, including VPR, only consider a single move type. In this paper, we propose a robust adaptive strategy based on Gibbs sampling to automatically select the "best" move types throughout the annealing procedure based on Gibbs sampling. By "best" we mean the move types that will yield the best results on average in a probabilistic sense.

Gibbs sampling is a special case of the Metropolis-Hastings algorithm, and thus an example of the Markov chain Monte-Carlo algorithm. Mathematically, it is an algorithm to generate a sample sequence from the joint probability distribution of two or more random variables. The purpose of such sequence generation is to approximate the joint distribution that can not be written in closed-form or too costly to compute directly. The key idea of Gibbs sampling is that, given a multivariate distribution, it is much simpler to sample from a conditional distribution than to integrate over a joint distribution. More specifically, suppose that a sample $\mathbb{X}$ is taken from a distribution depending on a parameter vector $\theta \in \Theta$ of length d with prior distribution $g(\theta_1, \dots, \theta_d)$. It may be that d is large and that numerical integration to find the marginal densities of the θ_i would be computationally expensive, or the form of $g(\theta_1, \dots, \theta_d)$ is too complicated to be evaluated. Then an alternative method of calculating the marginal densities according to Gibbs sampling is to create a Markov chain on the space Θ by repeating these two steps that define a reversible Markov chain with the desired invariant distribution g.

1) Pick a random index $1 \leq j \leq d$.
2) Pick a new value for θ_j according to $g(\theta_1, \dots, \theta_{j-1}, \cdot, \theta_{j+1}, \dots, \theta_d)$.

Based on the above scheme, we propose the following scheme to adaptively select effective move types. Let $m_i \in M$, $i = 0, 1, \cdots, n-1$, be one of n possible move types. Initially, we draw a move from M with equal probability $\frac{1}{n}$ among all move types. Throughout the annealing process, we keep track of acceptance rate a_i of each move type m_i. As annealing progresses, during each iteration, moves are drawn with the probability

$$p(m_i) = \frac{a_i}{\sum_{i=0}^{n-1} a_i}. \tag{3}$$

In this paper, we choose three move types m_1, m_2, and m_3 with different range limits $R_{\text{range}} = 2L/3, 4L/3, 2L$, respectively. R_{range} is measured as the Manhattan distance and L equals the number of clustered logic blocks (CLBs) along the X and Y directions of the target FPGA chip. Additionally, each move type m_i is associated with two types of counters $c_{\text{total},i}$ and $c_{\text{succ},i}$, which count the total number of move of type m_i performed and the total number of moves of type m_i accepted, respectively. At the beginning of the annealing, each of the three move types m_1, m_2, and m_3 are equally picked and their associated counters $c_{\text{total},1}$, $c_{\text{total},2}$, $c_{\text{total},3}$, $c_{\text{succ},1}$, $c_{\text{succ},2}$, and $c_{\text{succ},3}$ are set to 0. As the placement progresses, the success rate a_i of each move type s_i is updated as $\frac{c_{\text{succ},i}}{c_{\text{total},i}}$ and the probability to be picked for m_i will be determined by (3). To a large extent, our approach based on Gibbs sampling is similar to the range limiting functionality of VPR, detailed in [6]. However, our approach is much more general in that the move types under consideration are not limited to range differences. Instead, the set of move types can also include pipeline retiming and architecture-specific ones as discussed in [33]. Additionally, the range limit R_{limit} changes continuously according to a static formula, whereas in our Gibbs based approach, the range limit is adaptively but discretely determined.

B. Detecting Entrapments in Local Minima

Because stochastic tunneling is most effective when applied right after entrapments in local minima occur [20], [22], detecting local minima entrapment is crucial to the success of our proposed dynamically adaptive stochastic tunneling DAST approach. What makes such entrapment detection in stochastic tunneling particularly challenging is that, in general, the optimization procedure itself has no special knowledge about the underlying problem. As mentioned in Section III, simulated annealing typically alternates between "global" search at high temperature and "local" search at relatively low temperature. In other words, if we treat the objective function values at different iterations as a timing series signal, such a sequence is highly non-stationary, which makes traditional stationary signal processing techniques less effective. For example, one naive indicator for the evolving state of the DAST procedure would be the derivative $\frac{\Delta\text{score}}{\Delta\text{n}}$. Intuitively, the smaller the absolute value of $\frac{\Delta\text{score}}{\Delta\text{n}}$ is, the more likely no further progress beyond the current iteration can be achieved. This, however, depends on the function chosen and can therefore not serve as a generic measure. Our results using this indicator to detect local minima entrapments only resulted in very marginal improvements in the final optimization results.

To accurately detect the local minima entrapments and permit the detection/quantification of long-range correlations in the progressing of the proposed DAST algorithm, we opt to use a powerful statistical method called *DFA* [34]. As implied by its name, DFA was conceived as a method for detrending local variability in a sequence of events, and hence providing insight into long-term variations in the data sets. It is useful for analyzing time series that appear to be long-memory processes and was originally proposed as a technique for quantifying the nature of long-range correlations. Based on the DFA, we propose the following procedure to be used in the DAST algorithm.

1) Let $\mathbb{N}$ be the current iteration number,[1] t be the iteration index, and x_t be the annealing cost computed by the placement algorithm at the iteration t. Treat the history of x_t up to $\mathbb{N}$ as a bounded time series and compute the cumulative sum by $X_t = \sum_{i=1}^{t}(x_i - \langle x_i \rangle)$, where $\langle x_i \rangle$ is $\frac{\sum_{i=0}^{\mathbb{N}} m_i}{\mathbb{N}}$.
2) Divide X_t into time windows of length L samples, fit each segment of X_t with a straight line by minimizing the squared error $E^2 = \sum_{i=1}^{L}(X_i - ai - b)^2$ with respect to the slope and intercept parameters a, b. The initial value of L depends on the total iteration number $\mathbb{N}$. Our experience shows that $\lceil \mathbb{N}/L \rceil \approx 10$ is a good choice.
3) Calculate the root-mean-square deviation from the trend (or fluctuation) in each segment by
$$F(L) = \left[\frac{1}{L}\sum_{i=1}^{L}(X_i - ai - b)^2\right]^{\frac{1}{2}}.$$
4) Repeat Step 3 for different L, and finally a log-log graph of L against $F(L)$ is constructed to find $F(L) \propto L^{\alpha}$. To reduce the computation, in this paper, we choose four different Ls according to $\lceil \mathbb{N}/L \rceil \approx 10$, 20, 30, and 40, respectively. Finally, the scaling exponent α is calculated

[1]In DAST, we only perform local minima detection every 10 000 iterations.

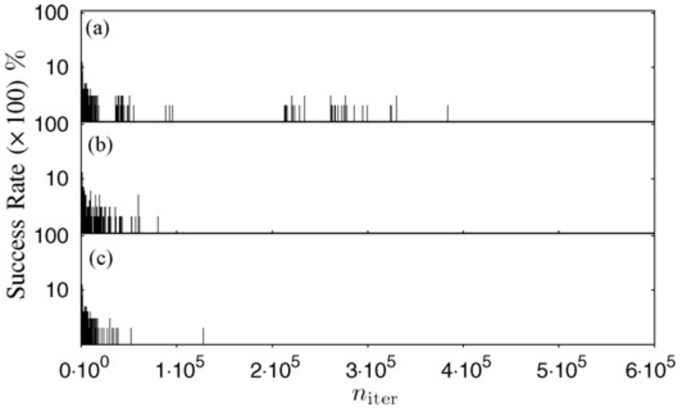


Fig. 4. Success rate comparison of annealing moves vs. iteration number. (a) Dynamically adaptative stochastic tunneling. (b) Static stochastic tunneling. (c) Conventional simulated annealing.

as the slope of a straight line fit to the log-log graph of L against $F(L)$ using least-squares.

According to DFA, the value of computed α strongly indicates the long-term stationarity of a time series. Normally, higher α means higher stationarity, i.e., the time series flattens out. In this paper, whenever α value exceeds 0.75, we claim that a local minima entrapment is detected. In other words, the annealing process is not making progress. The threshold value of 0.75 is empirically determined.

C. *Alternating Between Local Search and Tunneling*

Ideally, a good global optimization technique should be capable of avoiding local minima by itself. As stated in Sections II and III, both conventional simulated annealing and static stochastic tunneling suffer from the "freezing problem" due to low annealing temperature, although to different extent. Our numerical experiments and other studies [29] have shown that applying the stochastic tunneling technique throughout the annealing schedule can quickly degrade the effectiveness of local moves. Intuitively, as annealing progresses toward its final stage and its solution becomes close to the global minimum, the effect of the stochastic tunneling transformation results in a golf course-like solution space, i.e., the solution space in large is relatively flat but the minimum is hidden in a very localized region, therefore very hard to find. To circumvent this "golf-course" issue, instead of applying stochastic tunneling all the time, our proposed DAST algorithm alternates between a conventional simulated annealing and a "tunneling phase" once a local minima entrapment is detected by a DFA.

To illustrate the effectiveness of this alternating annealing schedule, we plot in Fig. 4(a)–(c) the success rate of annealing moves vs. iteration number for conventional simulated annealing, static stochastic tunneling, and the DAST algorithm, respectively. Success rate here is defined as the ratio between the number of accepted moves and each epoch of 100 moves. These results are obtained by placing and routing the largest MCNC design, `tseng`. Note an accepted move does not always produce a better solution. As shown in Fig. 4(b) and (c), the majority of successful moves are heavily concentrated in the early stage of annealing. After the first several thousands of moves, the success rate quickly declines to almost zero, which indicates entrapment in a local minimum. The fact that the success rate does not rebound even with a large number of additional iterations shows that, without external perturbation, it is hard to escape from this entrapment. In contrast, in Fig. 4(a), spikes of high success rates can be observed throughout the annealing process and clearly show the effect of escaping from local minima by stochastic tunneling.

D. DAST *Algorithm*

The key steps of our proposed DAST approach are listed in Algorithm 1, which differentiates it from the conventional simulated annealing method in several aspects. First, DFA is performed throughout the whole annealing schedule. Depending on detection of local minima entrapment, the objective function is evaluated differently. Second, different move types are considered as a whole and different move types are chosen according to the probability of success computed by a heuristic method based on Gibbs sampling, which is in turn determined by both the annealing history and the current topology of the solution space. Third, instead of using a pre-chosen value for the tunneling parameter γ, we use a simple adaptive technique, i.e., a dynamically choosing γ to ensure $\frac{E-E_0}{\gamma} \approx 0.05$. The value 0.05 is first suggested in [29] and is confirmed by our numerical experiments. Additionally, our experiments have shown that choosing a γ value dynamically is only effective when the annealing schedule alternates between conventional simulated annealing and stochastic tunneling. One plausible explanation is that in DAST, stochastic tunneling is only applied after local minima entrapment is detected and therefore only serves as an "annealing perturbation" on the otherwise conventional simulated annealing schedule.

V. Experimental Results and Analysis

A. *Experimental Approach*

An prototype annealing-based placer, called DAST, was developed as part of this paper. While DAST targets the same CAD flow as VPR, its simulated annealing algorithm is replaced with the proposed DAST algorithm detailed in Algorithm 1. Modified VPR software modules include the annealing scheduler, the temperature updating subroutine, and the cost evaluator.

The software implementation of DAST is largely based on the VPR5 package [35], specially designed to both support heterogeneous FPGAs and to provide an entire Verilog-to-routing CAD flow including the packing placement (VPACK) and routing. As illustrated in Fig. 5, the VPR5 flow starts with ODIN that provides elaboration of Verilog followed by logic synthesis with a specific version of the ABC tool [36] from UC Berkeley. Logic packing is performed with an updated version of TVPack that can pass heterogeneous blocks through untouched. Finally, placement and routing is performed with the latest version of VPR. More details about the VPR5 flow can be found in [37].

VPR5 uses an XML-based architecture file format in order to conveniently model hierarchy, which gives VPR the capability to implement circuits with hard blocks. We used the FPGA

Algorithm 1 Dynamically Adaptive Stochastic Tunneling Algorithm.

```
1: Generate an initial random solution;
2: Initialize $T$;
3: Initialize $m$ counters $c_i$, $i = 0, 1, \cdots, m-1$;
4: while $T > 0$ do
5:   while Thermal Equilibrium not reached do
6:     Perform multi-modal selection scheme on counters
       $c_{\text{total},i}$ and $c_{\text{succ},i}$, $i = 0, 1, \cdots, m-1$ to generate a
       random solution $s$;
7:     Compute DFA (Detrended Fluctuation Analysis);
8:     if Local entrapment detected then
9:       Using $\frac{E-E_0}{\gamma} \approx 0.05$ to adjust $\gamma$;
10:      Evaluate $s$ and obtain the new state $T$ by Computing $E$ with $e^{-\frac{E-E_0}{\gamma}}$;
11:    else
12:      Evaluate $s$ and obtain the new state $T$ by Computing $E$ with the original objective function;
13:    end if
14:    Evaluate $\Delta E$;
15:    if $\Delta E < 0$ then
16:      Update the current state and accept $s$;
17:    end if
18:    if $\Delta E \geq 0$ then
19:      Update the current state and accept $s$ with probability $p$;
20:    end if
21:  end while
22:  Update Temperature $T$;
23: end while
```

architecture file `k4-n4.xml` from `VPR5` with the number of LUTs per CLB set to 4 in order to resemble a Virtex II style architecture [38]. Using the CAD flow in Fig. 5, we successfully synthesized circuits with hard multipliers from Verilog through the ODIN tool, and then packed, placed and routed on a FPGA with 18×18 multipliers and CLBs of four 4-input LUTs, with $F_{c,in} = 0.15$, $F_{c,out} = 0.25$, $F_s = 3$, and segments of length $L = 4$. More detailed timing model of our FPGA architecture can be found at the iFAR website [39]. Instead of the more commonly used MCNC benchmark suite, this paper uses benchmark circuits provided by `VPR5` because they are more realistic and larger in size. Most of these circuits are obtained from OpenCores `http://www.opencores.org` and internal University of Toronto projects.

A good annealing schedule is essential to obtain high-quality solutions in a reasonable computation time with simulated annealing. To ensure our baseline competitive, we adopt the optimized annealing schedule proposed in [6], which leads to high-quality placements, and in which the annealing parameters automatically adjust to different cost functions and circuit sizes. We compute the initial temperature in a manner similar to [40]. Let N_{blocks} be the total number of logic blocks plus the number of I/O pads in a circuit. We first create a random placement of the circuit. Next, we perform N blocks moves (pairwise swaps) of logic blocks or I/O pads, and compute the standard deviation of the cost of these N different

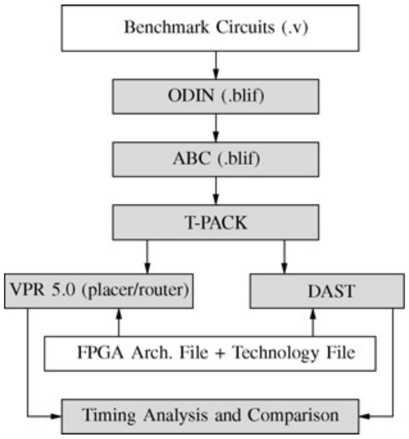


Fig. 5. CAD flow of performance comparison.

configurations. The initial temperature is set to 20 times this standard deviation, ensuring that initially virtually any move is accepted at the start of the anneal. As in [6], the default number of moves evaluated at each temperature is $10 \times N_{\text{blocks}}^{1.33}$.

Low-stress routing, i.e., allowing 15% wider routing channel width, was employed in all cases as this paper aimed to quantify the benefits of different types of annealing perturbations on the quality of placement results and not to measure the minimum architectural channel widths. To alleviate the effect of random seeds, each circuit from the test suite was run with five different seeds and all reported results are averaged over the five seeds used for each design.

B. Numerical Results

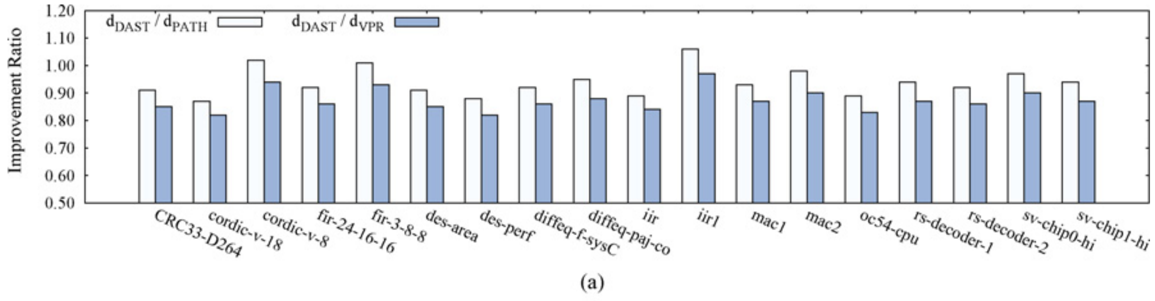
To illustrate the performance benefits of `DAST`, we compare it against both `VPR5`—a widely used academic placer [37] and `PATH`—a timing-driven placement scheme with net weighting developed in [7]. To make our performance comparison fair, we perform all numerical experiments using timing-driven placement in order to minimize critical-path delay. In both `VPR` and `DAST`, we use the timing cost function derived from `T-VPlace` [41], which is both wireability-driven (minimizing wiring requirements) and timing-driven. Because considering only wireability will often degrade timing performance, we follow the approach of `T-VPlace` and simultaneously consider critical path delay and wireability. The key innovation of `T-VPlace` is to define the auto-normalizing cost function ΔC as

$$\Delta C = \lambda \cdot \frac{\Delta C_{\text{timing}}}{C_{\text{timing,prev}}} + (1-\lambda) \cdot \frac{\Delta C_{\text{wire}}}{C_{\text{wire,prev}}}$$

where λ is a tradeoff variable that determines how much weight to give each cost component, ΔC_{timing} and ΔC_{wire} represent the cost change in timing and routability, and $C_{\text{timing,prev}}$ and $C_{\text{wire,prev}}$ denote previous cost components updated once every temperature. More details of timing computation and cost formula can be found in Section 3.2 of [41]. Our experiments suggest that $\lambda = 0.6$ yields the best results on average, which is close to the suggested value 0.5 in the study [41]. In the experiments using `PATH`, we use the net weighting function suggested in [7], which has relatively low complexity,

TABLE I
PERFORMANCE COMPARISON BETWEEN DAST, PATH, AND VPR

		Critical-Path Delay (ns)			Min. Req. # of Tracks			Placement Runtime (s)		
Circuit	# of CLBs	VPR	PATH	DAST	VPR	PATH	DAST	VPR	PATH	DAST
CRC33-D264	103	10.09	9.16	8.52	22	21	23	128	146	112
cordic-v-18	2948	6.24	5.41	5.12	111	109	117	242	293	194
cordic-v-8	556	4.43	4.53	4.16	54	51	56	310	325	297
fir-24-16-16	1553	11.62	10.67	9.92	83	80	88	540	612	477
fir-3-8-8	79	3.75	3.77	3.46	18	18	19	425	508	354
des-area	1426	4.09	3.72	3.48	81	85	85	236	322	161
des-perf	4457	4.86	4.27	4.00	142	139	143	675	817	551
diffeq-f-sysC	237	23.78	21.92	20.40	33	33	35	247	285	211
diffeq-paj-co	485	25.14	23.97	22.19	45	45	47	871	1035	732
iir	299	8.74	7.79	7.26	35	34	34	674	707	644
iir1	298	12.15	12.89	11.73	35	34	37	124	147	103
mac1	3044	9.93	9.21	8.57	110	105	110	456	552	364
mac2	10 373	12.73	12.44	11.48	213	221	215	246	276	218
oc54-cpu	2201	11.62	10.30	9.65	103	104	105	765	882	659
rs-decoder-1	1085	10.96	10.26	9.56	69	69	74	231	280	189
rs-decoder-2	1673	13.62	12.55	11.68	91	87	96	234	319	163
sv-chip0-hi	7039	5.94	5.75	5.31	186	180	181	178	194	164
sv-chip1-hi	37 703	3.42	3.20	2.97	179	164	192	1023	1093	953



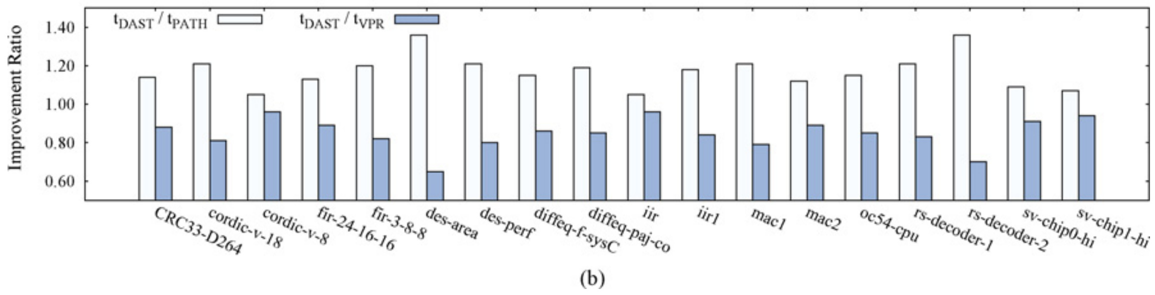


Fig. 6. Performance comparison between VPR, PATH, and DAST. (a) Performance improvements in critical-path delay d. (b) Performance improvements in runtime t.

strong flexibility, and easy implementation. The basic idea of PATH is to put a higher weight for nets that are more timing critical.

We use two metrics to compare the performance of DAST, PATH, and VPR: the total placement runtime and the final QoR. The QoR in this paper is measured by the critical path delay that includes the LB delays along the path. The QoR improvement in this paper is defined as the ratio of the critical-path delay for the same baseline FPGA but computed with the proposed DAST or PATH against the unmodified VPR. Results for all 18 benchmark circuits are listed in Table I. Note, as shown in Fig. 6(a) and (b) the improvements of DAST over VPR ranges from −5.29% to 10.97% for the critical path delay and from 4.43% to 32.35% for the runtime reduction, respectively. Similarly, the improvements of DAST over PATH range from −5.29% to 10.97% for the critical path delay and from 4.85% to 36.45% for the runtime reduction, respectively. Unfortunately, unlike runtime reduction, the improvements of DAST in critical-path delay are actually negative in several designs, but within 6%. To illustrate the routability performance in each approach, we also list in Table I the number of minimum required routing tracks $N_{\text{track,min}}$ in order to route each resulted placement by VPR, PATH, or DAST. For all 18 benchmarks we used, both PATH and DAST yield about the same $N_{\text{track,min}}$, which on average is about 3% lower than that of VPR.

C. Results Analysis

Given DAST's improvements to critical path delay and overall runtime, one might suspect that, just by changing

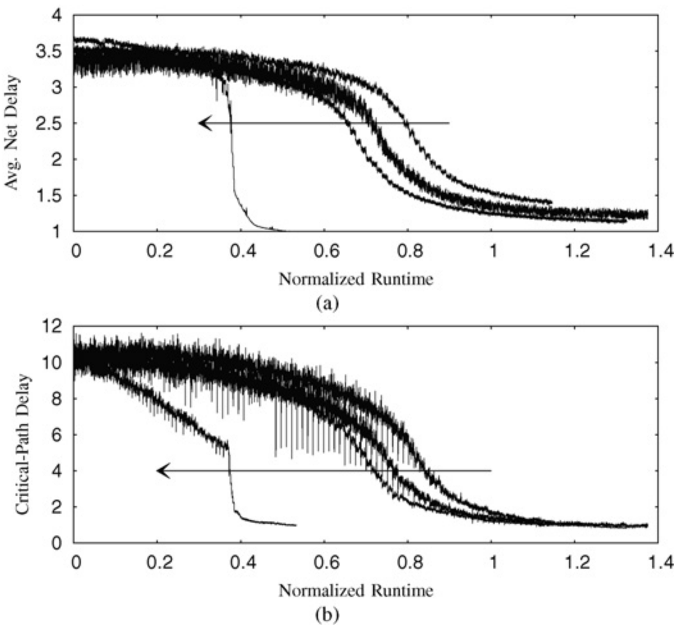


Fig. 7. Annealing history of VPR, PATH [7], SST, and DAST for sv-chip1-hi. In both (a) and (b), along the direction of the arrow, the four curves are in the order of VPR, SST, PATH, and DAST, respectively.

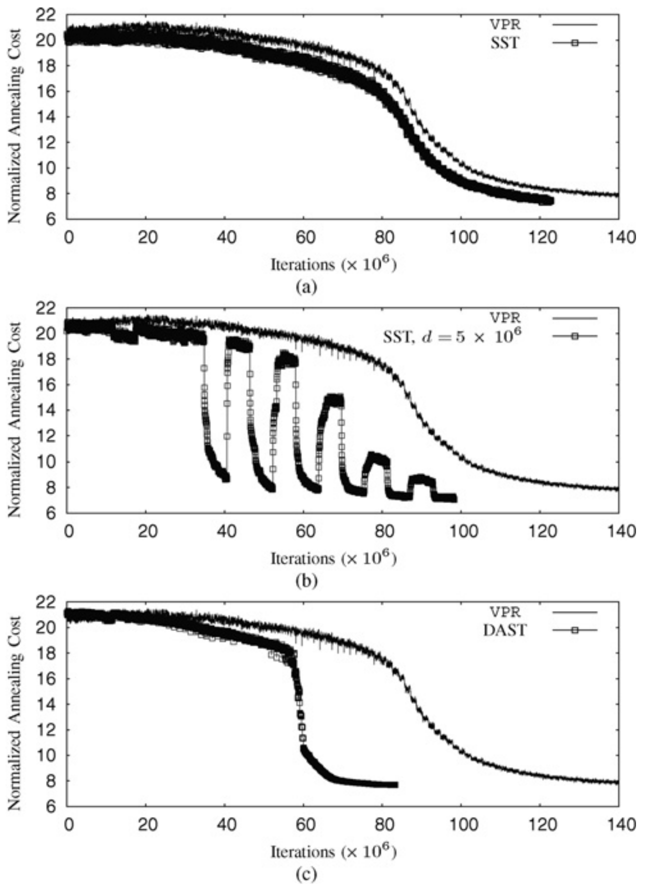


Fig. 8. Annealing history of sv-chip1-hi for stochastic tunneling with various alternating schedules. (a) Stochastic tunneling with no dynamic adaptation. (b) Stochastic tunneling with dynamic adaptation of alternating interval 5×10^6. (c) Dynamically adaptive stochastic tunneling.

the timing tradeoff parameter λ, VPR could achieve similar critical-path delay results. This is not the case. To validate this assertion, the timing tradeoff parameter λ was swept from 0.1 to 1.0 for generating placements with VPR. Our numerical results have shown that for the same amount of runtime, DAST consistently performed about 8% better in critical path delay than VPR on average. Additionally, note the key contribution of DAST have only to do with improving the simulated annealing algorithm that drives FPGA placement, therefore its success is orthogonal to the design of the cost function. To verify this, we compared the performance of DAST with that of VPR while using the same PATH-like net weighting cost function. Again, we observed the similar performance improvements as in Fig. 6(a) and (b). Specifically, for the 18 benchmark designs we used, the improvements of DAST over VPR range from −2.34% to 7.28% for the critical path delay and from 10.89% to 31.15% for the runtime reduction, respectively.

To further understand the performance improvement of DAST, we examine the annealing history of one benchmark—sv-chip1-hi. For easy visualization, we present in Fig. 7(a) and (b) the average net delay and critical-path delay for the benchmark sv-chip1-hi, plotted against the runtime, both of which are normalized against the delay and runtime of pure VPR—thus, a value of "1.0" on the y or x-axis roughly corresponds to the QoR or runtime achievable by an VPR annealer with an inner_num of 20 and an initial starting temperature of 20 times the measured standard deviation [6]. Care was taken to ensure that the annealing parameters were properly tuned for each of the runtime comparison points presented in this paper. Specifically, to alleviate the effect of random seeding, for each benchmark run, we run five times using different random seeds and take the average of all five results. Additionally, both VPR and DAST use the same inner number 20. In addition to comparing DAST with VPR, we also compare the overall QoR of DAST with the PATH with net-weighting scheme [7] and SST, where the annealing algorithm of VPR is replaced with the conventional stochastic tunneling.

As shown in Fig. 7(a) and (b), the DAST converges almost two times faster than the conventional VPR, while the critical-path delay achieved by DAST is about 8.3% shorter than that of VPR. Our results have shown that the net weighting algorithm for timing-driven placement [7] only achieves about 15% runtime reduction and 7% improvement in critical-path delay, respectively. In both Fig. 7(a) and (b), it can be seen that DAST can significantly accelerate the annealing; the alternating between conventional VPR and DAST is reflected in the abrupt slop changes of the DAST curves. There are actually five changes between VPR and DAST, but only the last one is obvious.

To illustrate the effectiveness of dynamically alternating between VPR and DAST, Fig. 8(a) and (b) presents the annealing history of various alternating schedules for the benchmark sv-chip1-hi. Specifically, Fig. 8(a) presents the results of stochastic tunneling without dynamically alternating, while Fig. 8(b) present the results of DAST with

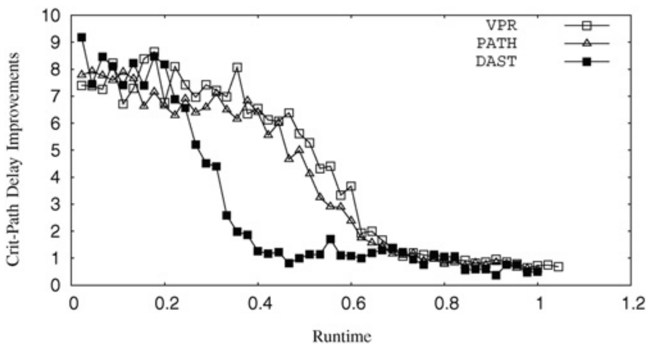


Fig. 9. Overall performance comparison between VPR, PATH [7], and DAST.

a fixed alternating schedule. As shown in these figures, by changing from the static stochastic tunneling to the fixed alternating interval of 5×10^6, the runtime changes from 1.23×10^8 to 1.00×10^8, clearly showing that the alternating schedule significantly impacts the convergence rate. Finally in Fig. 8(c), the dynamical adaptation is applied to the stochastic tunneling, which reduces the converging iteration number further down to about 8.5×10^7.

Finally, to overview the performance benefit of DAST over the conventional VPR, we placed and routed all 18 benchmarks using both approaches. For each benchmark design, we normalize both critical-path delay and runtime values from DAST against that of VPR. To make the comparison objective, for VPR, the starting temperature is chosen to be 20 times the measured standard deviation of random placement. Sixteen different combinations of random seeds and inner_nums are used and we pick the set that produces the best results as our reference for normalization. After obtaining 20 such performance curves, we use the geometric average to collapse them into one curve. As shown in Fig. 9, on average, DAST improves the runtime by approximately 30% and the critical-path delay by about 12%.

VI. Conclusion

This paper presented a DAST algorithm which improves the quality of FPGA placement while reducing the placement runtime. Three key ingredients of the DAST algorithm are: 1) a detection scheme for local minima entrapment based on DFA; 2) a dynamically adaptive annealing schedule based on stochastic tunneling; and 3) a heuristic to automatically select move types based on Gibbs sampling.

DAST can readily be incorporated into existing FPGA placement software and benefit from other research work specifically targeting FPGA placement improvement. We hope that this paper will motivate the FPGA community to systematically evaluate the many newly developed and promising optimization techniques, such as stochastic tunneling, parallel tempering, and particle swarm optimization. As a versatile algorithm, DAST may also be applied to other global optimization problems, such as protein folding, and has the potential to yield significant performance improvements there as well.

Acknowledgment

The authors would like to thank Y. Ma and B. Burke for the help with software implementation.

References

[1] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” in *Proc. ACM/SIGDA 10th Int. Symp. FPGA*, 2006, pp. 21–30.

[2] M. Lin and A. El Gamal, “TORCH: A design tool for routing channel segmentation in FPGAs,” in *Proc. Int. Symp. FPGA*, 2008, pp. 131–138.

[3] Y. Sankar and J. Rose, “Trading quality for compile time: Ultrafast placement for FPGAs,” in *Proc. ACM/SIGDA 7th Int. Symp. FPGA*, 1999, pp. 157–166.

[4] C. Mulpuri and S. Hauck, “Runtime and quality tradeoffs in FPGA placement and routing,” in *Proc. ACM/SIGDA 9th Int. Symp. FPGA*, 2001, pp. 29–36.

[5] J. Cong, M. Romesis, and M. Xie, “Optimality and stability study of timing-driven placement algorithms,” in *Proc. IEEE/ACM ICCAD*, 2003, p. 472.

[6] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *Proc. 7th Int. Workshop FPLA*, 1997, pp. 213–222.

[7] T. T. Kong, “A novel net weighting algorithm for timing-driven placement,” in *Proc. IEEE/ACM ICCAD*, 2002, pp. 172–176.

[8] Y. Zhang, “Progress and challenges in protein structure prediction,” *Current Opin. Structural Biol.*, vol. 18, no. 3, pp. 342–348, 2008.

[9] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[10] K. Vorwerk, A. Kennings, and J. Greene, “Improving simulated annealing-based FPGA placement with directed moves,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 2, pp. 179–192, Feb. 2009.

[11] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, “Placement and routing tools for the triptych FPGA,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[12] P. Gopalakrishnan, X. Li, and L. Pileggi, “Architecture-aware FPGA placement using metric embedding,” in *Proc. 43rd ACM/IEEE Des. Autom. Conf.*, Jul. 2006, pp. 460–465.

[13] R. Tessier, “Fast placement approaches for FPGAs,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 2, pp. 284–305, 2002.

[14] P. Maidee, C. Ababei, and K. Bazargan, “Fast timing-driven partitioning-based placement for island style FPGAs,” in *Proc. 40th Annu. DAC*, 2003, pp. 598–603.

[15] L. Ingber, “Adaptive simulated annealing (ASA): Lessons learned,” *CoRR*, vol. cs.MS/0001018, no. 25, pp. 33–54, 2000.

[16] K. Eguro and S. Hauck, “Enhancing timing-driven FPGA placement for pipelined netlists,” in *Proc. 45th ACM/IEEE DAC*, Jun. 2008, pp. 34–37.

[17] D. Abramson, “A very high speed architecture for simulated annealing,” *Computer*, vol. 25, no. 5, pp. 27–36, 1992.

[18] R. Schneider and R. Weiss, “Hardware support for simulated annealing and tabu search,” in *Proc. Workshop Biologically Inspired Solutions to Parallel Process. Problems Int. Parallel Distribute Process. Symp.*, 2000, pp. 660–667.

[19] M. G. Wrighton and A. M. DeHon, “Hardware-assisted simulated annealing with application for fast FPGA placement,” in *Proc. ACM/SIGDA 11th Int. Symp. FPGA*, 2003, pp. 33–42.

[20] A. Schug, T. Herges, and W. Wenzel, “Reproducible protein folding with the stochastic tunneling method,” *Phys. Rev. Lett.*, vol. 91, p. 158102, Oct. 2003.

[21] A. J. Tolley and M. Wyman, “Stochastic tunneling in DBI inflation,” *ArXiv e-prints*, Sep. 2008.

[22] A. Baumketner, H. Shimizu, M. Isobe, and Y. Hiwatari, “Stochastic tunneling minimization by molecular dynamics: An application to heteropolymer models,” *Phys. A Stat. Mech. its Applicat.*, vol. 310, pp. 139–150, Jul. 2002.

[23] K. Hamacher, “Adaptation in stochastic tunneling global optimization of complex potential energy landscapes,” *Europhys. Lett.*, vol. 74, no. 6, pp. 944–950, 2006.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[25] V. Granville, M. Krivanek, and J.-P. Rasson, “Simulated annealing: A proof of convergence,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 6, pp. 652–656, Jun. 1994.

[26] J. de Vicente, J. Lanchares, and R. Hermida, "Placement by thermodynamic simulated annealing," *Phys. Lett. A*, vol. 317, nos. 5–6, pp. 415–423, 2003.

[27] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0*. 1991.

[28] Z. Li and H. A. Scheraga, "Monte Carlo-minimization approach to the multiple-minima problem in protein folding," *Proc. Natl. Acad. Sci. USA*, vol. 84, no. 19, pp. 6611–6615, Oct. 1987.

[29] W. Wenzel and K. Hamacher, "Stochastic tunneling approach for global minimization of complex potential energy landscapes," *Phys. Rev. Lett.*, vol. 82, pp. 3003–3007, Apr. 1999.

[30] K. Hamacher and W. Wenzel, "Scaling behavior of stochastic minimization algorithms in a perfect funnel landscape," *Phys. Rev. E*, vol. 59, pp. 938–941, Jan. 1999.

[31] U. H. E. Hansmann and L. T. Wille, "Global optimization by energy landscape paving," *Phys. Rev. Lett.*, vol. 88, p. 068105, Jan. 2002.

[32] I. P. Schagen, "An adaptive system for global optimization," *IMA J. Math. Control Info.*, vol. 3, no. 4, pp. 283–292, 1986.

[33] K. Eguro, S. Hauck, and A. Sharma, "Architecture-adaptive range limit windowing for simulated annealing FPGA placement," in *Proc. 42nd Annu. DAC*, 2005, pp. 439–444.

[34] C.-K. Peng, S. V. Buldyrev, S. Havlin, M. Simons, H. E. Stanley, and A. L. Goldberger, "Mosaic organization of DNA nucleotides," *Phys. Rev. E*, vol. 49, pp. 1685–1689, Feb. 1994.

[35] University of Toronto. (2009). *VPR and T-VPack 5.0.2* [Online]. Available: http://www.eecg.utoronto.ca/vpr

[36] UC Berkeley. (2009). *A System for Sequential Synthesis and Verification* [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc

[37] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proc. ACM/SIGDA Int. Symp. FPGA*, 2009, pp. 133–142.

[38] *Virtex-II Complete Datasheet (All Four Modules)*, Xilinx, Inc., San Jose, CA, Nov. 2007, pp. 1–28.

[39] University of Toronto. (2009). *iFAR: Intelligent FPGA Architecture Repository* [Online]. Available: http://www.utoronto.ca/vpr/architectures

[40] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA: Kluwer, 1990.

[41] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. ACM/SIGDA 8th Int. Symp. FPGA*, 2000, pp. 203–213.

Mingjie Lin (M'08) received the B.S. degree in engineering from Xi'an Jiaotong University, Xi'an, China, in 1996, the M.S. degree in mechanical engineering and the M.S. degree in electrical engineering from Clemson University, Clemson, SC, in 2001, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2008.

From 2008 to 2009, he was with a FPGA startup, Tabula, Inc., Santa Clara, CA, as a Senior Engineer. Missing the atmosphere of academic research, he returned to academia at the beginning of 2009. Since 2009, he has been working as a Post-Doctoral Scholar with the Department of Electrical Engineering and Computer Science, University of California, Berkeley. He is currently an Assistant Professor of Electrical Engineering and Computer Sciences with the University of Central Florida, Orlando. His previous research has involved very large scale integration reconfigurable array architecture, bio-inspired/neuromorphic arrays, and monolithically stacked 3D-IC. His current research interests include exploring novel ways to construct scalable computing machines with high performance and low power consumption. To this end, his research activities have spanned across computer architecture/compiler, reconfigurable computing, integrated circuits, and system design.

John Wawrzynek (M'85) received the B.S. degree in electrical engineering from the State University of New York at Buffalo, Buffalo, the M.S. and Ph.D. degrees in computer science from the California Institute of Technology, Pasadena, and the M.S. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana.

He is currently a Professor of Electrical Engineering and Computer Sciences with the University of California, Berkeley. He is a Faculty Scientist with the Lawrence Berkeley National Laboratory, NERSC Division, Berkeley, and is a Co-Director with the Berkeley Wireless Research Center, Berkeley. He currently teaches courses in digital design, computer architecture, very large scale integration system design, and reconfigurable computing. He was a Co-Founder of Andes Networks, Mountain View, CA, a company specializing in the design and manufacturing of accelerators for network security, and of BEECube, Inc., Fremont, CA, a company specializing in reconfigurable computing systems. His current research interests include reconfigurable computing and computer architecture.