EVOLUTIONARY HEURISTICS FOR MULTIOBJECTIVE VLSI NETLIST BI-PARTITIONING

Sadiq M. Sait[1] , Aiman H. El-Maleh[2] , and Raslan H. Al-Abaji[3]

Computer Engineering Department, KFUPM
Dhahran 31261, Saudi Arabia
E-mail: {sadiq,aimane,raslan}@ccse.kfupm.edu.sa

ABSTRACT

The problem of partitioning appears in several areas ranging from VLSI, parallel programming, to molecular biology. The interest in finding an optimal partitioning especially in VLSI, and has been a hot issue in recent years. In VLSI circuit partitioning, the problem of obtaining a minimum cut was of prime importance. Furthermore, with current trends partitioning has become a multi-objective problem, where power, delay and area in addition to minimum cut, need to be optimized. In this paper we employ two iterative heuristics for the optimization of VLSI Netlist Bi-Partitioning. These heuristics are based on Genetic Algorithms (GAs) and Tabu Search (TS) [sadiq et al., 1999] respectively. Fuzzy rules are incorporated in order to design a multiobjective cost function. Both the techniques are applied to ISCAS-85/89 benchmark circuits and experimental results are reported and compared.

Keywords: *Genetic, Tabu, VLSI, Fuzzy, Partitioning.*

الملخص

مشكلة التقسيم تظهر في مختلف المجالات العلمية كالأنظمة المتكاملة (VLSI) والبرمجة المتوازية وحتى علم الأحياء الجزيئي. الإهتمام بإيجاد القسمة المثلى خاصة في مجال الأنظمة المتكاملة كان موضوعا هاما في الزمن القريب. في الأنظمة المتكاملة مشكلة ايجاد القطع المثالي حظيت على الأهمية القصوى. ومع التطورات الحديثة اصبحت القسمة ذات طابع متعدد الأهداف يعالج مشكلة الطاقة والوقت والمساحة بالأضافة االى تقليل وصلات المقطع. في هذا البحث نستخدم خوارزميان تكراريين هما الخوارزم الجيني (GA) والبحث الممنوع (TS). تم استخدام قواعد مبهمة لتصميم دالة تجمع الأهداف كافة. تم اختبار كلتا التقنيتين والمقارنة بينهما من حيث الأداء والسرعة.

1. INTRODUCTION

VLSI circuit design can be performed with a view of achieving different objectives. Until the beginning of this decade, two main objectives of VLSI Partitioning circuit design were in focus: one was the minimization of cutset and the other was the improvement of timing performance. A large number of efforts targeting either one (especially cutset) or both of the above objectives are reported in the literature [Sadiq et al. 1995, Shahookar et al. 1991]. Although quite a reasonable number of techniques aiming at low power objective are proposed for all phases in physical design including partitioning of circuit, floorplanning, placement and routing [Sadiq et al. 1995], the power consumption of the circuit was not of main concern while trying to optimize the above two objectives. As different techniques are applicable and have been reported in [Pedram., 1995] at different steps of the VLSI design process, few performance-driven partitioning techniques at physical level design exist in literature. Furthermore another compelling reason for the desire of low power consumption is the increasing density of VLSI circuits. Therfore, the need of a system which incorporate all the three aspects of the design process (delay, cut, power) is increasing and to our knowledge, no effort has been reported that targets the optimization of the three objectives simultaneously. This fact provides a significant motivation for the present work.

For the partitioning phase, two low-power oriented techniques based on Simulated Annealing (SA) algorithm have recently been presented in [Choi et al., 1999]. Algorithms targeting low power are proposed in [Vaishnav et al., 1999, Lawler et al. 1969]. A circuit partitioning algorithm under path delay constraints is proposed in [Tetsushi et al., 1998]. VLSI design is a complex process and is carried out at several abstraction levels [Sadiq et al., 1995]. The problem of power optimization can be addressed at higher levels as well as at lower levels e.g., physical level [Pedram., 1995, Devadas et al., 1995]. In this work, we address the above problem in the Partitioning step at the physical level. Two iterative approaches based on genetic algorithm (GA) and tabu search (TS) respectively, are presented for the multiobjective optimization of Partitioning. This paper is organized as follows: In the next section, we formulate our problem and cost function. Section 3 presents our approaches, and then experimental results are reported and discussed in section 4.

2. PROBLEM AND COST FUNCTION MODELING

In this section, we formulate our problem and the cost function used in our optimization process.

2.1 Problem Formulation

We are addressing the problem of VLSI Netlist partitioning with the objectives of optimizing power consumption, timing performance (delay), and cut-set while considering the Balance constraint (same as area constraint because we assume unit area for every gate.). Formally, the problem can be stated as follows:

Top

Given a set of modules $V = \{v_1, v_2, .., v_n\}$, the purpose of partitioning is to assign the modules to a specified number of clusters *k* satisfying prescribed properties.

Definition :

A *k*-way partitioning $P^k = \{C_1, C_2, .., C_n\}$ consists of *k* clusters (subsets of *V*) $C_1, C_2, .., C_k$ such that $C_1 \cup C_2 ... \cup C_k = V$ and $C_i \cap C_j = \varnothing$ for *i=1…k* and *j=1…k*. If *k = 2* we refer to P^2 as bi-partitioning.

In general, the circuit can have multi-pin connections (nets) apart from two-pin, therefore it is better to be described by a hypergraph especially when we want to minimize the net cut. A hypergraph is *H(V,E)* where *V* is a set of nodes and *E* is a set of hyperedges. Node $v_i \in V$ corresponds to an element (e.g., a gate) in the circuit, and hyperedge $e_i \in E$ corresponds to a net in the circuit. Hyperedge $e_i \in E$ consists of the signal source node *S(e_i)* and a set of destination nodes *D(e_i)* and $e_i = (\{S(e_i)\}, D(e_i))$. The signal source node *S(e_i)* of the net e_i corresponds to the output of a gate and the set of destination nodes *D(e_i)* corresponds to the inputs of the gates. Given a hypergraph *H(V,E)* with $E = \{e_1, e_2, ..., e_m\}$ being the set of signal nets, each net is a subset of *V* containing the modules that the net connects, and we assume that for each $e \in E$, such that $|e| \geq 2$. The equivalence between netlists and hypergraphs is exact if each net has at most one pin on any module. The modules in *e* may also be called the pins of e. Moreover, a two-way partitioning of a set of nodes *V* is to determine two subsets V_A and V_B such that $V_A \cup V_B = V$ and $V_A \cap V_B = \{\varnothing\}$. Before listing the objectives and constraints, we state some assumptions: (i) The logic level design (or netlist) of the circuit is available, (ii) A set of critical paths and switching probabilities of gates are available before the application of our proposed approaches, (iii) The circuit is represented in form of a hypergraph. Our task is to divide *V* into *2* subsets (blocks) in such a way that the objective functions are optimized, subject to some constraints.

2.2 Cost Function

Now we formulate cost functions for our three objectives (cut, power, delay) and for the Balance constraint.

Cutsize The set of hyperedges cut by a cluster *C* is given by $E(C) = \{e \in E : 0 < |e \cap C| < |e|\}$ i.e., $e \in E(C)$ if at least one, but not all, of the pins of *e* are in *C*. The set of nets cut by a partitioning solution P^K can be expressed as $E(P^k) = \bigcup_{i=1}^{k} E(C_i)$ or equivalently $E(P^k) = \{e \in E \mid \exists u, v \in e, h \neq l, u \in C_h \; and \quad v \in C_l\}$. We say that the cutsize of P^k is $|E(P^k)|$.

The cost function can also be written as follows :

$$f = \sum_{e \in \psi} w(e) \tag{1}$$

where $\psi \subset E$ denotes the set of off-chip wires. The weight *w(e)* on the edge *e* represents the cost of wiring the corresponding connection as an external wire. If all weights equal one, the cost function becomes simpler:

$$f = |\psi| \tag{2}$$

where $|\psi|$ denotes the cardinality of the set ψ.

Delay In order to deal with a signal path, we decompose a hypergraph into directed edges $e_k = (S(e_i), w)$ for $e_i \in E$ and $w \in D(e_i)$. Let the graph which consists of a set of nodes V and a set of decomposed directed edges E be the directed graph $G' = (V, E)$. A signal path is represented by an alternating sequence of nodes and directed edges $v_1, e_1, v_2, e_2, ... v_{k-1}, e_{k-1}, v_k$, where $e_l = (v_l, v_{l+1}) \cdot (1 \le l \le k-1)$ and $v_i \ne v_j$, $i \ge 1, j \le k, i \ne j$. The path from node V_i to node V_j is denoted by P_{ij}. Nodes which are included in the path P_{ij} are defined as $V(P_{ij})$. A path-cut number of path P_{ij}, denoted $ncut(P_{ij})$, is the number of nets cut which are included in the path P_{ij}. The general delay model in which gate delay is $d(v)$ and constant inter-chip wire delay d_c » $d(v)$ are considered; the d_c is actually due to the off-chip capacitance denoted as C_{off}. Let the delay of node $v_i \in V$ be $d(v_i)$ and the delay of net $e_k \in E$ which is cut be d_c. Given a partition $\Phi : (V_A; V_B)$, the path delay $d(P_{ij})$ between nodes v_i and v_j is the sum of the node delays $d(v_i) \in V(p_{ij})$ and the delay of nets which are cut, that is :

$$d(p_{ij}) = \sum_{v_k \in V(p_{ij})} d(v_k) + d_c \cdot ncut(p_{ij}) \tag{3}$$

Power The average dynamic power consumed by a CMOS logic gate in a synchronous circuit is given by:

$$P_i^{average} = 0.5 \frac{V_{dd}^2}{T_{cycle}} C_i^{load} N_i \tag{4}$$

where C_i^{load} is the load capacitance, V_{dd} is the supply voltage, T_{cycle} is the global clock period, and N_i is the number of gate output transitions per clock cycle. N_i is calculated using the symbolic simulation technique of [Ghosh et al., 1992] under a zero delay model. C_i^{load} in Eqn. 4 consists of two components: C_i^{basic} which accounts for the load capacitances driven by

Top

a gate before circuit partitioning, and the extra load C_i^{extra} which accounts for the additional load capacitance due to the external connections of the net after circuit partitioning. Then, the total power dissipation of circuit ζ is:

$$P_\zeta = \beta \frac{V_{dd}^2}{T_{cycle}} \sum_{i \in \zeta} (C_i^{basic} + C_i^{extra}) N_i \tag{5}$$

where β is a constant that depends on technology. When a circuit partitioning corresponds to a physical partitioning, C_i^{extra} of a gate that is driving an external net is much larger than C_i^{basic}. In this case, the power model given in Eqn. 5 can be further simplified by assuming that the power dissipation contribution due to variations of C_i^{basic} under different partitioning solutions is negligible. Furthermore, considering that the fixed overhead capacitance for an external net is dominant within C_i^{extra}, it can be assumed that C_i^{extra} is identical for each net. This leads to the following objective function [Vaishnav et al., 1999].

$$O\zeta = \sum_{i \in \zeta_v} N_i \tag{6}$$

where ζ_v corresponds to the set of visible gates, i.e., the set of gates that drive a load external to the partition.

Area or Balance constraint If we assume that the area of all cells identical then the problem reduces to balancing the two partitions in term of the number of cells. The balance constraint is given in Eqn 7.

$$\frac{|\beta_1 - \beta_2|}{\phi} \le \alpha \tag{7}$$

where β_i is the number of cells in partition i, ϕ is the total number of cells in the circuit, α is the tolerance which is equal zero in case of perfect balance.

Overall Fuzzy Cost Function Since we are optimizing three objectives simultaneously, we need to have a cost function that represents the effect of all three objectives in form of a single quantity. We propose the use of fuzzy logic to integrate these multiple, possibly conflicting objectives into a scalar cost function. Fuzzy logic allows us to describe the objectives in terms of linguistic variables. Then, fuzzy rules are used to find the overall cost of a placement solution. In this work, we have used the following fuzzy rule:

Top

IF a solution has
SMALL cut-set **AND**
LOW power consumption **AND**
SHORT delay **AND**
Within acceptable Balance
THEN *it is an GOOD solution.*

The above rule is translated to and-like OWA (order weighted averaging) fuzzy operator [Yager , 1998] and the membership μ (x) of a solution x in fuzzy set GOOD solution is given as:

$$\mu^c_{pdc}(x) = \beta^c \cdot \min(\mu^c_p(x), \mu^c_d(x), \mu^c_c(x)) + (1-\beta^c)\cdot\frac{1}{3}\sum_{j=p,d,c}\mu^c_j(x) \qquad (8)$$

$$\mu^c(x) = \min(\mu^c_{pdc}, \mu^c_b(x)) \qquad (9)$$

where $\mu^c(x)$ is the membership of solution x in fuzzy set of acceptable solutions, μ^c_{pdc} is the membership in fuzzy set of "acceptable power AND acceptable delay AND acceptable cutset", whereas $\mu^c_j(x)$ for j = {p,d,c,b}, are the membership values in the fuzzy sets within acceptable power, within acceptable delay, within acceptable cutset and within acceptable balance respectively. β^c is constant in the range [0,1]; the superscript c represents the "cost ". $\mu^c(x)$ is used as the aggregating function. The solution that results in maximum value of $\mu^c(x)$ is reported as the best solution found by the search heuristic. The membership functions for fuzzy sets LOW power consumption, SHORT delay, Small cut-set, are shown in Figure 1. We can vary the preference of an objective *j* in overall membership function by changing the value of goal vector g_j, wich represents the relative acceptable limits for each objective where $g_j \geq 1.0$.

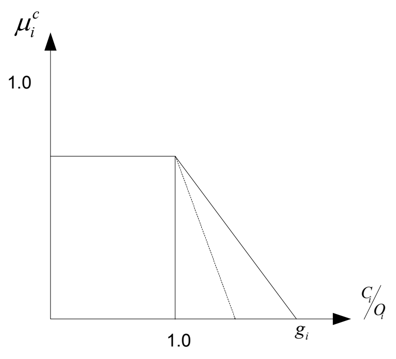

Figure 1 Membership functions

Top

3. PROPOSED APPROACHES

In this section, we describe the implementation details of the proposed approaches. First, we discuss the details of Genetic Algorithm for Multiobjective Partitioning and then, we briefly describe the implementation of Tabu Search (TS).

3.1 Genetic Algorithm (GA) For Timing and Low Power Driven Partitioning

There have been many efforts involving the application of GA to the VLSI Partitioning problem. Earliest application of GA for the Min-Cut Bisection was proposed by Akley in [Ackley, 1987], later Bui and Moon utilized GAs for graph bisection [Bui et al., 1994]. Recently GA are becoming famous specially in Multi-objective optimization problems. GA is an elegant search technique that emulates the process of natural evolution as a means of progressing towards the optimal solution. GA's Start from a population of solutions and creates new generations by means of crossover and mutations, the good characteristics of selected good parents are supposed to be preserved and the survival of the fittest should guarantee the improvement of the solution.

3.1.1 Chromosome Encoding and Initial Solution

GA uses an encoded representation of solution in the form of a string made up of symbols called genes. The string of genes is called *chromosome*. One way to represent the partitioning problem (as seen in Figure 2 (a) is to use *group-number encoding* where the j^{th} integer $i_j \in 1,\dots,k$ indicates the group number assigned to object j. This representation scheme creates a possibility of applying standard operators. The second representation scheme is in Figure 2 (b). Here, the solution of the partitioning problem is encoded as *n+k-1* strings of distinct integer numbers. This representation scheme leads to 100% feasible solutions but requires more computation time due to the complexity of the unary operator involved. In our implementation we used the first representation.

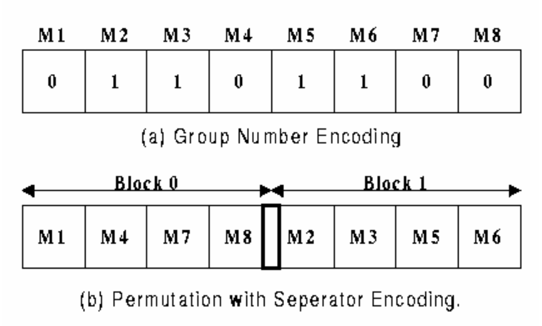

Figure 2 Representation schemes.

The algorithm starts with a set of initial solutions called *population* that is generated randomly or taken from the results of a constructive algorithm. When generating the random initial solution it is preferred that it is within the bounds of the balance constraint.

3.1.2 Fitness Evaluation

For addressing a multi-objective optimization problem to minimize three mutually conflicting objectives, a measure is needed which can quantify the overall quality of a solution with respect to all three objectives collectively.

Fuzzy membership functions and fuzzy rules are used for evaluating the fitness of a solution. A fitness value between 0 and 1 is assigned to each solution. The fitness value of a chromosome is its membership value $\mu(x)$ in the fuzzy set of acceptable solution. This membership is computed using equation 9.

3.1.3 Crossover and Mutation

In each iteration (known as generation in GA terminology), all the individual chromosomes in the population are evaluated using a fitness function. Then in the selection step, two of the above chromosomes at a time are selected from the population. The individuals having higher fitness values are more likely to be selected. After the selection step, different operators namely crossover, mutation act on the selected individuals for evolving new individuals called offsprings. These genetic operators are described below. One important genetic operator is crossover. It is applied on two individuals that were selected in the selection step earlier to generate an offspring. The generated offspring inherits some characteristics from both its parents in a way similar to natural evolution. There are different crossover operators namely simple(single point), order, partially mapped, and cycle. The simple crossover for instance, works by choosing a random cut point in both parent chromosomes (the cut point should be the same in both parents) and generating the offspring by combining the segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut [Sadiq et al., 1999]. For description of other crossover operators [Shahookar et al., 1991, Sadiq et al., 1999, Cohoon et al. 1987]. Increasing the number of crossover points is known to be multi-point crossover. For the group number encoding two-point and three-point crossovers are favored. The crossover operator however may produce children that violate the balance constraint. These can be treated in two ways, either discarding them by giving them a fitness value of zero, or fixing them using some constructive heuristic and this is too much time overhead. In our implementation we used single point crossover.

The ***Mutation*** operator is used to introduce new random information in the population. It is usually applied after the crossover operator. It helps in producing some variations in the solutions so that the search does not get trapped in local minima. An example of mutation operation is the swapping of two randomly selected genes of a chromosome. The importance

Top

of this operation is that it can introduce a desired characteristic in the solution that could not be introduced by the application of the crossover operator alone. However, mutation is applied with a low rate so that GA does not turn into a memory-less search process [Bui et al., 1994.] Two mutation variations are used in our implementation:

- Random selection of a cell and swapping its partition.
- Choosing randomly two cells one from each partition and swapping them.

3.1.4 Selection

Individuals for the next population are selected based on the *elitist-random selection (ernd).* $\frac{N_p}{2}$ (N_p is the population size) best chromosomes are selected and the remaining $\frac{N_p}{2}$ are selected randomly. Based on experimental results this scheme offers better choice than other schemes, because it provides balance between greediness and randomness.

The quality of the solution obtained from GA is dependent on the choice of certain parameters such as population size, crossover and mutation rates and also the type of crossover used. The selection of values for these parameters is problem specific and so there are no hard and fast rules for this purpose. The choice of these parameters is left to the conception and intuition of the person applying GA to a specific problem.

3.2 Tabu Search Approach

In this sub-section, we describe our TS implementation very briefly.

Tabu search starts from an initial feasible solution and carries out its search by making a sequence of random moves or perturbations. A tabu list is maintained that stores the attributes of a number of previous moves. This list prevents bringing the search process back to already visited states. In each iteration, a subset of neighbor solutions is generated by making a certain number of moves and the best move (the move that resulted in the best solution) is accepted, provided it is not in the tabu list. Otherwise, if the said move is in the tabu list, the best solution is checked against an *aspiration criterion* and if satisfied, the move is accepted. Thus, the aspiration criterion can override the tabu list restrictions. It is desirable in certain conditions to accept a move even if it is in the tabu list, because it may take the search into a new region due to the effect of intermediate moves.

The solution encoding and initialization steps are similar to those described above for GA. In each iteration, we generate a number of neighbor solutions by making perturbations as follows: two cells are selected randomly, then their locations are interchanged. The number of neighbor solutions generated in each is dependent on circuit size. The characteristic of the move that we keep in tabu list is the indices of the cells involved in interchange. The size of tabu list is taken also depending on the circuit size i.e. 10% of the total number of cells. We

Top

have used short term memory element in our TS implementation. The aspiration criterion used is that if current best solution is the best seen so far i.e. better than the global best, then it is accepted and tabu restriction are overridden.

4. EXPERIMENTAL RESULTS AND DISCUSSION GA VERSUS TS.

The results obtained from GA and TS are compared in terms of overall quality of best solution and run time in Table 1. In this table, P represents the cost due to power, that is the sum of the switching probabilities of all the cut nets, it has no unit since switching probability has no unit, D is the delay of the most critical path in pico seconds *(ps)*, $\mu(x)$ is the membership value, T is the total run time, and Best is the execution time in seconds for reaching the best solution. In the case of TS 10,000 iterations are run, also for GA the stopping criterion is 10,000 generations.

Table 1 Comparison between costs of the best solutions generated by GA and TS

Circuit	Genetic Algoritm						Tabu Search					
	D (Ps)	Cut	P(SP)	$\mu(x)$	T (S)	Best(S)	D(Ps)	Cut	P(SP)	$\mu(x)$	T (S)	Best(S)
S298	233	19	1013	0.79	123	43	197	24	926	0.809	62	21
S386	356	36	1529	0.75	163	151	386	30	1426	0.729	82	77
S641	1043	45	2355	0.83	1868	1540	889	59	2281	0.852	939	818
S832	444	45	3034	0.68	289	276	446	50	2731	0.682	148	80
S953	526	96	2916	0.69	618	182	466	99	2518	0.734	313	225
S1196	396	123	5443	0.76	375	373	301	106	4920	0.801	184	134
S1238	475	127	5713	0.72	397	365	408	79	4597	0.752	187	160
S1488	571	104	5648	0.71	1238	1183	528	98	5529	0.724	616	405
S1494	614	102	5474	0.70	1228	1040	585	101	5339	0.702	616	427
S2081	302	26	787	0.73	94	32	225	17	770	0.785	47	16
S3330	571	299	10358	0.75	2096	2074	533	295	10298	0.79	1078	994
S5378	587	573	18437	0.74	2687	2686	590	430	16527	0.79	1338	1100
S9234	1313	1090	38149	0.72	5963	5949	1052	918	34055	0.81	2992	2821
S13207	1399	1683	45611	0.74	8098	8097	843	1332	41114	0.785	4001	3690
S15850	1820	2183	51747	0.74	10214	10206	1411	1671	47480	0.831	5131	5130

The results shown are the best case results obtained after the tuning of various algorithmic parameters of GA and TS (only one time for all circuits). In the case of GA the population size is 10, the crossover used is simple with a probability equal to 0.99. In case of TS, the size of neighborhood is also 10, while tabu list size is chosen to be 0.1 size of the circuit. From the results, it is clear that TS performed better than GA for most of the circuits in terms of quality of best solution as well as run time.

In terms of quality of solution TS consistently performs better, and the advantage of TS over GA gets more consistent when the size of the circuit gets bigger. Also in term of the execution time of GA increases significantly with the increase in circuit complexity. The higher execution time of GA can be attributed to its parallel nature i.e. a population of solutions is to be processed in each generation. Figures 3 and Figures 4 show the trend of solution's (a) cutset, (b) delay, (c) power, (d) balance, (e) average fitness, (f) best fitness for GA and TS respectively, in case of circuit s12307. It is clear from the shown plots that TS achieves a membership that is better than that reached by GA. the best solution for GA is found after 8097 seconds and has 0.74 fitness value while for TS was found after 3690 seconds and have a higher fitness value of 0.79.

5. CONCLUSION

In this paper, two multiobjective optimization iterative algorithms namely GA and TS for VLSI Partitioning were proposed. Fuzzy logic is used to integrate the objectives namely power, delay, cutset and balance into a scalar cost value. It is observed that TS out performs GA in terms of final solution costs and execution time.

ACKNOWLEDGMENT

The authors thank King Fahd University of Petroleum Minerals, Dhahran, Saudi Arabia, for support, unde project # : {COE/ITERAT E/221}

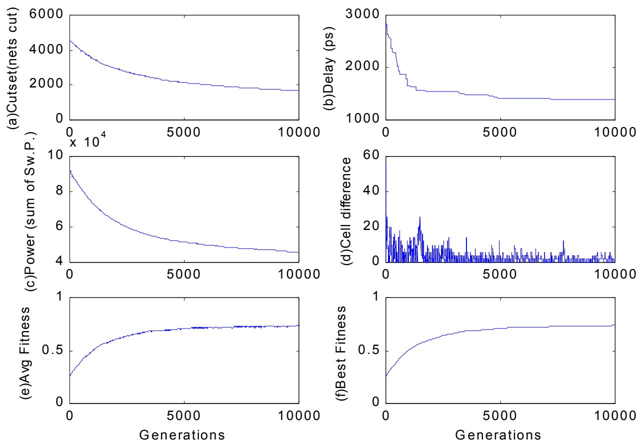

Figure 3 Performance of GA for the circuit s13207

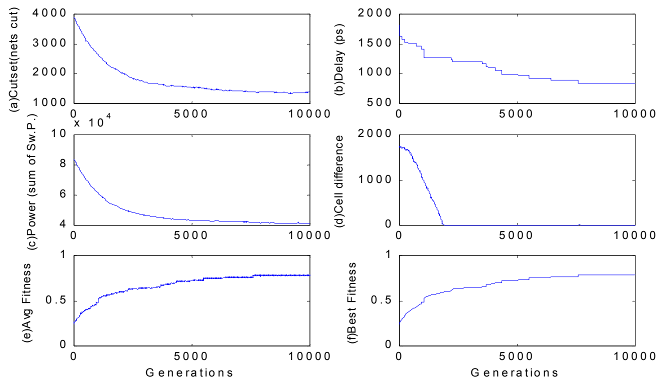

Figure 4 Performance of TS for the circuit S13207

REFERENCES

1. Ackley, 1987, D. H. Ackley. *A Connectionist Machine For Genetic Hillclimbing.* Kluwer, 1987.
2. Bui et al., 1994, T.Bui and B. R. Moon. *A Genetic Algorithm for a Special Class of the Quadratic Assignment Problem.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16:99-116, 1994.
3. Choi et al., 1999, I.S. Choi and S.Y. Hwang. *Circuit Partitioning Algorithm for Low-Power Design Under Area Constraints Using Simulated Annealing.* IEE Proceedings Circuits Devices Systems, 146(1):8-15, February 1999.
4. Cohoon et al. 1987, J.P. Cohoon and W. D. Paris. *Genetic Placement.* IEEE Transactions on CAD, pages 956-964, 1987.
5. Devadas et al., 1995, Srinivas Devadas and Sharad Malik. *A Survey of Optimization Techniques Targeting Low Power VLSI Citcuits.* 32nd ACM/IEEE Design Automation Conference, 1995.
6. Ghosh et al., 1992, Abrajit Ghosh, Srinivas Devadas, Kurt Keutzer, and Jacob White. *Estimation of average switching activity in combinational and sequential circuits.* Design Automation Conference, pages 253-259, 1992.
7. Lawler et al. 1969, E. Lawler, K. Levitt, and J. Turne. *Module Clustering to minimize delay in digital networks.* IEEE trans. On Computer-Aided Design, 47-57, 1969.
8. Pedram., 1995, Massoud Pedram. *CAD for low power: Status and Promising Directions.* IEEE International Symposium on VLSI Technology, Systems and Applications, pages 331-336, 1995.
9. Sadiq et al., 1995, Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation. Theory and Practice.* McGraw-Hill Book company, Europe, 1995.
10. Sadiq et al., 1999, Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, California December 1999.
11. Shahookar et al., 1991, K. Shahookar and P. Mazumder. *VLSI Cell Placement Techniques.* ACM Computing Surveys, 2(23):143-220, June 1991.

12. Tetsushi et al., 1998, Jun'ichiro minami tetsushi and Koide Shin'inchi wakabayashi. *A Circuit Partitioning Algorithm Under Path Delay Constraints.* IEEE, 1998.
13. Vaishnav et al., 1999, H. Vaishnav and M. Pedram. *Delay optimal partitioning targeting low-power VLSI circuits.* IEEE Trans. On Computer Aided Design, 18(6):298-301, June 1999.
14. Yager , 1998, Ronald R. Yager. *On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision Making.* IEEE Transaction on Systems, MAN, and Cybernetics, 18(1), January 1988.

Top