ELSEVIER

Microprocessing and Microprogramming 40 (1994) 537–552

Microprocessing and Microprogramming

# Genetic algorithm for embedding a complete graph in a hypercube with a VLSI application

R. Chandrasekharam[a,*], V.V. Vinod[b], S. Subramanian[b]

[a] *Department of Computer Science & Engineering, Regional Engineering College, Warangal 506004, India*

[b] *Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur 721302, India*

**Abstract**

The embedding of a complete graph in a minimum sized hypercube is an important problem which models the classical state encoding problem of Finite State Machines (FSMs). As this problem is an NP-hard optimization problem, acceptable final solutions are generally obtained by employing heuristic methods or Simulated Annealing (SA). In this paper the efficacy of a Genetic Algorithm (GA) for this problem is studied. This study includes a comparison of three different crossover methods of GA along with their implementation details and their suitability for this embedding problem. The experimental results on a number of MCNC benchmark FSMs indicate the superiority of GA in finding a better (near optimal) solution than a heuristic solution. These results experimentally establish the time efficiency of GA over SA for this embedding problem.

*Keywords:* Genetic algorithm; Hypercube; VLSI

## 1. Introduction

The general graph embedding problem is defined as follows. Let $G_g(V_g, E_g)$ and $G_h(V_h, E_h)$ be the guest and host graphs described by their vertex sets $V_g$, $V_h$ and edge sets $E_g$, $E_h$ respectively. It is assumed that the guest graph has at most as many vertices as that of the host graph. Let $u$, $v$ be any two vertices of $G_g$ and an embedding function $F$: $V_g \rightarrow V_h$ be such that $F(u) \neq F(v)$ if $u \neq v$. With respect to the embedding two factors namely *dilation factor* and *expansion factor* are defined. The term dilation $(u, v)$ for $u \neq v$ is defined as

$$\text{dilation } (u, v) = \frac{d_h(F(u), F(v))}{d_g(u, v)},$$

where $d_h$ and $d_g$ are distances in $G_g$ and $G_h$ respectively. The distance between two nodes is the minimum path length between the two nodes. The

* Corresponding author. Fax: 91 8712 76547.


*SSDI* 0165-6074(94)00015-3

dilation factor $D_f$ is given by

$D_f = \max\{\text{dilation } (u, v)\}$ for all $u \neq v$.

The expansion factor $E_f$ is defined as $|V_h|/|V_g|$ and in general embedding is attempted with expansion factor $E_f \geqslant 1$ only. The embedding problem is to obtain a one-to-one embedding function $F$ such that $D_f$ is minimized. It may be observed that $D_f = 1$ means that the embedding preserves the adjacency properties of $G_g$ in $G_h$. This dilation factor essentially represents the largest of the ratio of the distances in the guest and host graphs. Hence, when dilation minimization is attempted for an embedding problem that models a real world problem, it results in the optimization of some objective with reference to that problem. The importance of the embedding problem can be explained by the real world problems modelled by it. Some such problems are briefly given below.

We encounter different embedding problem instances while porting some algorithms designed on one architecture onto another architecture [14]. In case of such embedding instances the vertices corresponds to the processors, and edges to the communication links. The dilation minimization in this case minimizes the inter processor communication delay, there by resulting in better time efficiency for the ported algorithm.

Embedding has been used to model the problem of processor allocation in a distributed system [10]. In this case a number of tasks are to be handled by different processors available in a distributed system. The problem of deciding which processor is to do which task is modeled as embedding problem. Dilation minimization results in minimization of inter-task communication overhead.

The problem of placing processors on a printed circuit board which form a hypercube architecture is addressed in [13]. The objective is minimization of the length of the longest wire used for interconnection of a pair of processors. This problem is modeled as embedding a hypercube in a mesh. A method of embedding is given for which the lower bound on the dilation is obtained.

Embedding a binary tree into a square grid [7] has been pointed out to be an important requirement of many VLSl circuits. In [7] using a building block U-tree it has been shown that the problem of determining whether a unit length (dilation) embedding of a tree in a square grid is possible, is a NP-complete problem.

The above mentioned instances of embedding are examples in which the host and guest graphs are different types. The general embedding problem in which the guest and host graphs are any arbitrary graphs is known to be NP-complete [12]. Imposing a restriction of host being a hypercube is of practical importance, since many parallel computers are being made available with hypercube architecture. Attempting problem solving on such machines would lead to the embedding with the above restriction. But, even this instance of embedding has been shown to be NP-complete [12]. In this work this specific instance of embedding is addressed since it models the important problem of encoding the states of a Finite State Machine. Formulation of this embedding instance and its equivalence with state encoding problem are presented in the following section.

## 2. Formulation of the specific embedding problem

The specific embedding we consider is the one in which the host is a minimum sized hypercube. A $k$-dimensional hypercube is a graph having $2^k$ vertices labeled from 0 to $2^k - 1$, and in which two vertices are joined by an edge is their labels (as binary numbers) differ in one bit position only. In this formulation we consider the guest graph as a complete graph in which there is an edge between every pair of vertices. Such a graph of $n$ vertices is described by a symmetric matrix $a(i, j)$ where each $a(i, j)$ is the weight associated with the edge between

vertex $i$ and vertex $j$. It may be noted that this description restricts to only undirected graphs. In case of embedding arbitrary graphs into a hypercube, the same description may be used with edge weight made zero if there is no edge in the guest graph. It may be observed that the edge weight of an edge in the guest graph corresponds to a problem specific quantity (such as the communication cost) and the problem is modeled by the embedding.

Let the vertices of the guest graph be labeled by $\{V_{g1}, V_{g2}, \cdots V_{gn}\}$. Since the embedding is attempted with expansion factor $E_f \geqslant 1$, the size of the hypercube is assumed to be $k$ such that $k = \lceil \log_2 n \rceil$. This corresponds to the embedding into a minimum sized hypercube. The problem of embedding the complete graph $G_g$ of size $n$ into a $k$-cube with $k = \lceil \log_2 n \rceil$ is now formulated as determining a mapping function $F: V_g \rightarrow V_h$. The embedding is done with an objective of minimising the cost function given by

$$\text{Embedding cost} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} a(i,j) * d_h(C_i, C_j) \qquad (1)$$

where $a(i, j)$ is the weight of the edge between nodes $V_{gi}$, $V_{gj}$. In other words, each node of the guest graph is encoded with a $k$-bit binary number by this embedding. With such encoding, let the codes given to the nodes $V_{gi}$, $V_{gj}$ be $C_i$ and $C_j$ respectively. The distance $d_h(V_{gi}, V_{gj})$ represents the dilation itself since the distance $d_g(V_{gi}, V_{gj})$ is unity as $G_g$ is a complete graph. The minimization of the embedding cost given by Eq. (1) may be observed to be equivalent to dilation minimization.

This embedding problem is explained with an example of guest graph with 5 nodes with edge weights as given in Fig. 1. Since guest graph has 5 nodes the host graph is a hypercube with 8 nodes. Each node of host graph is labeled with a 3 bit binary number. Let the nodes of guest graph be indicated by A, B, C, D, E. Consider the three

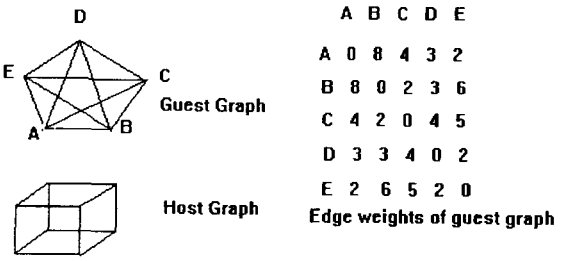



Fig. 1. Example embedding.

encodings corresponding to the embeddings denoted by EBD-1, EBD-2, and EBD-3.

EBD-1 = (A-000, B-001, C-010, D-011, E-100)

EBD-2 = (A-111, B-110, C-100, D-000, E-001)

EBD-3 = (A-001, B-000, C-101, D-010, E-100)

The cost of these three embeddings are 59, 71, and 56 respectively. The embedding problem is to determine the encoding that results in the minimum cost. It may be easily observed that the encoding obtained by reversing the three bit code assigned to each node will have the same cost. Hence, multiple optimal solutions may exist for a specific embedding problem.

The above mentioned specific embedding formulation models the problem of encoding the states of a FSM. Consider an FSM with $n$ states. The behaviour of such FSM is described by its state transition table. Each row of such table contains a list of primary inputs, a present state, a next state, and a list of outputs. With the specified input combination the FSM transits from the present state to the next state and asserts the specified outputs. Thus each row describes a transition of the FSM. From the description of all the transitions of the FSM given in its state transition table, a static estimation of the *encoding affinity* between all pairs of states can be computed. The *encoding affinity* between

two states is defined as the advantage gained by encoding these two states with minimum distant codes.

Let $a(i, j)$ denote the encoding affinity between the states $s_i$ and $s_j$. The corresponding matrix with entries $a(i, j)$ may be treated as the description of a complete graph. In this complete graph each vertex corresponds to a state of the FSM. The resulting embedding of the above formulation ends up in mapping each state $s_i$ of the FSM to a vertex of the hypercube labeled by the $k$-bit binary number $C_i$. When such embedding is done with an objective of minimizing the cost function given by Eq. (1) it is equivalent to finding the encoding of the states of the FSM with minimum encoding cost. The distance $d_h(C_i, C_j)$ denotes the Hamming distance between $C_i$ and $C_j$. As the cost of embedding depends on $a(i, j)$ also, determination of these encoding affinities from the FSM description becomes another important part of the state assignment problem. Many researchers [1–5, 8, 17] have taken different approaches to come out with encoding of the states of FSM. Some of the early methods [8] employed algebraic methods based on partition theory. The later approaches have been in the lines of the above mentioned graph embedding instance.

A common feature of all the state assignment algorithms such as MUSTANG [4] and NOVA [17] is that they all make static estimations of the encoding affinity between states, based on the literal savings. These literal savings are obtained by the logic minimizers that are generally employed after the state assignment. Thus a proper modeling of the targeted logic circuit, and the various boolean and algebraic techniques that are employed in the logic minimizers influence the literal saving cost estimates. For example literal savings due to boolean merge and consensus operations cannot be taken into account unless the encodings differ by unit distance. A comprehensive picture of literal saving estimates is presented by the authors of MUSE [5]. A closed form expression for the literal saving is obtained in terms of the FSM's symbolic cubes (one row of the state transition table is a symbolic cube) containing two parts. One part corresponds to the encoding affinity of the present state and the other to the next state. The part of present state encoding affinity is independent of the targeted structure of the logic circuit of the FSM. The approximations made by other methods like MUSTANG while computing this part of encoding affinity have been pointed out by the authors of MUSE. A better cost estimate for state assignment (encoding affinity) is presented by them which takes into account the present state as well as the next state encoding affinity.

The above mentioned methods have mainly addressed the issue of computing encoding affinity, so that the eventual realization of the FSM circuit is in minimum area. It may be pointed out here that *whichever way the encoding affinity is computed, the later step of determining the codes assigned to the symbolic states* is very important. With a given encoding affinity, this encoding problem turns out to be the problem of embedding a complete graph in a hypercube. This embedding instance is shown to be NP-complete and hence, we find only greedy heuristic methods or Simulated Annealing being employed. In our work the efficacy of GA for this embedding problem is addressed. In the following section all these three approaches namely the greedy, SA, and GA approaches for this specific embedding instance are presented in detail.

## 3. Solution methods

### 3.1. The Greedy methods

This method is based on the adjacency properties of the vertices in the hypercube. Since the label of

this vertex is a $k$ bit binary number, for every vertex there are $k$ vertices with unit distance. This property motivates the selection of a cluster of $k+1$ nodes with a central node $n_c$ and $k$ nodes around it in the guest graph for embedding at a time. Iteratively such clusters are identified and assigned with minimum distance codes. In each iteration the center of the cluster is deleted from the guest graph. A pseudocode given below explains the procedure.

**Procedure Greedy**
begin
while $G_g$ is not empty do
{
Select the central node $n_c$ and the cluster nodes $n_i$ such that $\sum a(c, i)$ is maximum
Assign $n_c$ and all the other $n_i$ possibly minimum distance codes.
Delete $n_c$ such that $G_g$ becomes $G_g - n_c$
}
end {Greedy}

This algorithm has been proposed by S. Devadas and its optimality property has been discussed in his paper on MUSTANG algorithm [4]. One may observe that this method may end up with an encoding which is suboptimal. It is because in every iteration we encounter some states to be encoded with possible minimum distant codes. It is possible that we have many choices in the early iterations and less or no choices in later iterations. Hence, the resultant encoding may turn out to be suboptimal. This can be illustrated by the following example.

Consider the example embedding given in the Fig. 1. The first cluster can be identified with center at node B and other nodes being A, D, and E. In the first iteration let the codes assigned to these four nodes be B-000, A-001, D-010, and E-100. After deleting the center of the cluster, we get the next cluster with center at C and nodes being A, D, and E. The code to be assigned to node C to be unidistant to the codes of A, D, and E. It may be noted that there is no code available which satisfies this requirement, as it is assigned to the node B. Among the available codes there are multiple choices which are unidistant to any two nodes in the cluster. Thus a proper choice of code for C determines the cost of encoding. If C is encoded with 101 the cost is 56 and if C is encoded with 110 the cost is 52. This example illustrates the possibility of the greedy approach leading to a suboptimal solution. Even though optimality is not ensured by this method, it is very fast and shown to have a worst case time complexity of $O(n^2 \log n)$.

### 3.2. *Simulated annealing*

This is a probablistic hill climbing relaxation algorithm proposed by Kirkpatrick et al. [11] as a general tool for solving hard optimization problems of NP-complete class. The method emulates the chemical process of annealing. In terms of optimization the energy of the object corresponds to the cost function. Starting from a random initial solution, in every iteration a new solution is obtained by a suitable perturbation. If the new solution is at least as good as the initial solution, then the new solution becomes the current solution. If not the new solution is accepted as the current solution with a probability $P$ given by

$$P = \exp(\delta c / T)$$

where $\delta c$ is the cost difference and $T$ is the temperature. The canonical form SA is given in Procedure SA as follows.

**Procedure SA**
begin
Initialize a random current solution $S$
Initialize the temperature as $T_0$

```
While (temperature ≠ 0) do
    begin
    for (some number of perturbations) do
    begin
        obtain the new solution S_n by perturbation of the current solution S
        compute the difference cost δc
        compute the acceptance probability P
        compute a random number R in the range 0.1
        if R ⩽ P new solution becomes the current solution
    end {for}
    update (temperature = α * temperature)
    end {while}
end {Procedure SA}
```

From the canonical form given above it can be observed that the final solution obtained by SA method essentially depends on the starting solution and the cooling schedule employed for updating the temperature. Any random initial solution can be easily generated for the embedding problem. An integer string of length $n$ would suffice as the solution. The binary code of the integer in the position $i$ is the code assigned to the vertex $i$ of the complete graph. Exchange of the codes assigned to two vertices or changing the code assigned to a vertex to some other unassigned code can be employed for the perturbation. A probabilistic selection of any of these perturbations may also be employed. The number of perturbations per temperature is determined experimentally. However, the choice must ensure some minimum number of down hill moves.

Though SA offers a general paradigm for solving hard optimization problems, it is observed to be critically dependent on the initial solution, the number of perturbations per temperature and the temperature schedule. The required slow cooling schedule may cause SA to become prohibitively slow for some hard problems.

### *3.3. Genetic algorithm*

GA has been proposed as a paradigm modeled on the lines of adaptation in biological systems. The genetic structure of biological organisms goes through many changes in a number of generations due to the genetic operators like crossover and mutation. In the process, a population of species gets enriched in certain genetic order over number of generations. The outcome of this is generally a population well adapted to the environment. Holland [9] proposed that a simulation of such adaptation process under controlled conditions can be an efficient approach for solving complex optimization problems. The pseudo-code for Procedure GA given below describes the canonical form of genetic algorithm.

**Procedure GA**
begin

1. Initialize the GA parameters.
2. Initialize a population of candidate solution strings.
3. Evaluate the objective function value for all solutions.
4. Evaluate the fitness value for all solutions.
5. Select the parent strings as per a chosen reproductive plan.
6. Perform operations and produce new offspring.
7. Evaluate the objective function values for the offspring.
8. Evaluate the fitness of the offspring.
9. Select the next generation population.
10. If the processing of generations still remains, then go to 5.
11. Select the best fit solution in the current population as final solution.

end {Procedure GA}.

From the Procedure GA, one may identify the components of GA to be

(1) A chromosomal representation of solution to the problem,
(2) A way to create some random initial population of the solutions,
(3) An evaluation function that plays the role of environment, rating the solutions in terms of their fitness,
(4) Genetic operators that alter the genetic structure of the solutions and
(5) Values for the GA parameters namely the population size, the number of generations, crossover rate, and the mutation rate.

These components of GA for the embedding problem are explained in the following section.

## 4. GA components

### 4.1. Solution representation

The embedding problem requires the mapping of one node of $G_g$ on to a node of $G_h$ to be expressed as a solution. An integer string of length $n$ would suffice for this purpose. The same representation holds good for the SA method also. Hence, we choose an integer string for solution in which the binary code corresponding to the integer in the position $i$ is the code assigned to the state $i$. Such a gene string solution is a natural requirement for the effectiveness of the adaptive search by GA.

### 4.2. Objective function and fitness of a solution

The objective of the assignment is to minimize the cost of encoding (embedding) given by Eq. (1). This cost can be computed for any given solution. The cost of a solution in a population of one generation is the basis to determine the fitness of that solution. The lowest cost solution is the best fit solution and the highest cost solution is the least fit solution. All other solutions that are processed in a single generation thus can be ordered relatively according to their fitness. Any solution with a better cost than that of the best fit solution pushed into the current population would change the fitness of each of the solutions in that population. Consequently improvement in fitness corresponds to lower cost.

### 4.3. Initial population

The FSM considered may have $n$ states which are less than the available $2^k$ codes. In such a case seed groups as suggested by Mitra [15] may be formed. An example of 5 states and 8 codes illustrates this. In this case the seed groups may be (0, 7) (1, 6) (2, 5) (3) (4). Thus, all possible solutions contain one code from each seed group. Such seed groupings would restrict the total search space. But, considering the flat nature of the objective function it has been observed that the approximation by seed groupings would not seriously effect the area of the targetted logic circuit of the FSM. These permissible codes of seed groupings are arranged in a seed table and a random permutation of one column of the seed table gives a valid solution. Using any permutation generation scheme iteratively the required initial population of solutions can be obtained. Such population is expected to have a good measure of diversity for an efficient search by GA. The diversity in case of this problem is a measure by which the gene structure differs. In other words, an equal distribution of all permissible codes for any single state is desirable and this ensures good diversity in the initial population. Hence, the initial population is generated with uniform distribution of all possible codes.

### 4.4. Crossover and mutation operations

The purpose of crossover is to obtain the offspring solution that inherits the genetic features of

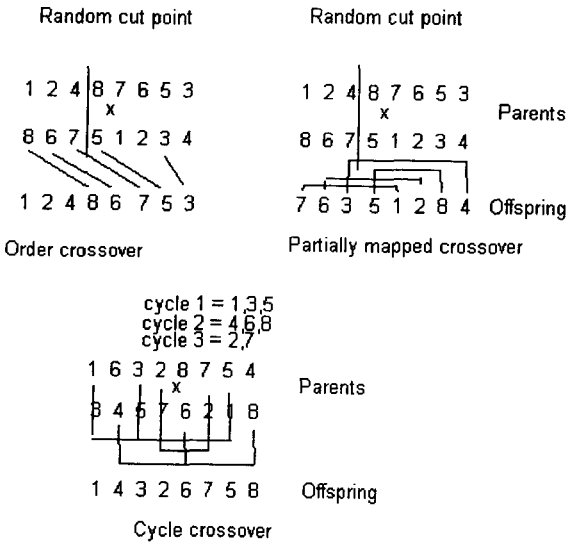



Fig. 2. Different crossover operations when the gene string is a permutation.

both parent solutions. Three types of crossover operations are reported in the literature [16] and these operations are illustrated in Fig. 2.

*Order crossover.* In this method the offspring inherits all genes from parent 1 up to the random cut point. Since the remaining genes are to be given distinct allele values, the offspring can be constructed by observing an order for these allele values in the parent 2. All genes that are present to the right of the cut point are given the allele values in the order they appeared in the parent 2.

*Partially mapped crossover.* All the genes of parent 1 are copied into the offspring first. A random cut point is chosen as in the case of order crossover. The segments to the right of the cut point in both the parents are considered for a partial mapping of the allele values to be exchanged inthe parent 1 to generate the offspring. The allele values of the genes in each position to the right of the cut point in both parents are noted. These are exchanged in the offspring. Care is taken not to exchange any gene with a specific allele value if it is already exchanged once. It may be observed that partially mapped crossover results in an offspring that preserves large number of allele values from parent 1 in the left cut segment, and from parent 2 in the right cut segment.

*Cycles crossover.* In this case every allele value of the gene in the offspring is inherited from one of the parents in its position and hence, results in no conflicts. We start with the gene with a specific value in the first position of parent 1 and copy it into the offspring. Consider the gene (possibly with some other allele value) in the same position of parent 2. Since it cannot be inherited in this position of the offspring, it is searched for in parent 1. When its position is found, this gene from parent 1 is copied into the offspring at this position. Since the allele value of the gene of parent 2 in this new position cannot be inherited by the offspring, once again the search may be conducted. This cycle of search and copy is continued until the cycle is complete. It may be noted that the cycle completes when the allele value of the gene in parent 2 becomes same as that of the first gene inherited by the offspring in the cycle. The process of inheritance is switched to parent 2 for the next cycle. The cycles are alternated between the two parents until all allele values are assigned to different genes.

*Mutation.* In the process of search by GA it is possible that all solutions in a population of one generation may turn out to be same. In such a case, the search is impaired since crossover cannot explore any more new solutions. GA gets trapped in a local minimum under these conditions. Mutation i.e. altering the genetic structure of few solutions by a small amount, is employed for surmounting this local optimum. However, the mutation level is kept low. Random selection of codes of few states

(decided by the mutation rate) and changing to new values is chosen as mutation operation.

### 4.5. Selection policies

Parents are selected based on their fitness in a conventional GA. With a view of saving the time taken for parent selection, in our work we adopted a policy of parents being completely at random, but with a restriction of no solution reproduces twice in a generation. The offspring replace the parents and form the next generation population in a conventional GA. With such strategy it is likely that a current best solution may not survive to the next generation. To enforce the elitism in which the current best solution survives into the next generation, we adopted a policy of replacing the worst o-number of solutions in the current population by the o-number of offspring that are generated. This policy ensures the survival of every new solution generated by the crossover operation for at least one generation. Such a policy is experimentally observed to yield a good efficiency for GA, since it permits maintaining a good amount of diversity in the population.

The above mentioned GA components explain the implementation aspect of GA. However, the question 'How GA works' still remains to be answered. Also the suitability of GA for the embedding problem needs to be explained. These are considered in the following section.

## 5. 'How GA works' – A qualitative picture

In the process of adaptation by GA, the successive rearrangements of smaller building blocks over a number of generations causes the construction of optimal or near optimal solution in the current population. In this context the similarities between the solution strings in a population become important. Holland introduced the framework of *Schema*

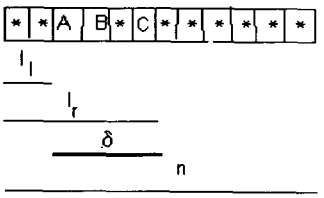



Fig. 3. Schema example.

as a tool to explain the building block concept. The Schema is a similarity template describing a subset of strings with same genes at some string positions. In other words Schema is a pattern matching tool. Schema can be viewed as a partial solution also. For example a schema $H$ illustrated in Fig. 3 expressed as $**A*BC******$ has the specified values $A$, $B$, and $C$ for three genes and nine unspecified genes indicated by $*$ in those positions. A number of such Schema can be defined, and a set of such partial solutions are referred to as Schemata. The population of solutions may be viewed as a collective representation of Schemata.

In the process of the artificial adaptation the population gets enriched in some schemata and gets diminished in some alleles. To understand which Schema grows in the population over the generations, we need to consider the properties of that Schema. The important properties or attributes of any Schema are its order and the defining length. The order of a Schema $H$ denoted by $O(H)$ is defined as the number of specified positions in the Schema $H$. The defining length $\delta(H)$ is defined as the length of the string between the first and last specified genes in the Schema $H$. It may be observed that for the given example Schema $O(H)$ is 3 and $\delta(H)$ is 4. As pointed out earlier the Schema may be viewed as a partial solution structure, or a building block of the solution. Schemata and their properties give a basic means for analyzing the GA method of arriving at optimal or near optimal solution from successive juxtapositions of

a number of low order and short defining length Schemata.

The process of adaptation by GA in the population and the eventual construction of optimal or near optimal solution is primarily dependent on the effect of reproduction, crossover and mutation on the survival of better fit Schema in successive generations. A mathematical framework explaining these effects is given below. Let the membership of Schema $H$ in the population $A$ at a time $t$ be denoted by $m(H,t)$. First we consider the effect of reproduction.

*Reproduction.* A solution participates in the process of reproduction depending on its fitness which is the method of natural selection. This means that a candidate solution in the population $A_i(t)$ gets selected with a probability $P_r$ where

$$P_r = f_i / \sum f_j$$

in which $f_i$ is the fitness of the $i$th solution. The membership of the Schema $H$ in two consecutive generations is thus given by

$$m(H,t+1) = m(H,t) * N * f(H) / \sum f_i$$

where $f(H)$ is the average fitness of the solutions containing the Schema $H$ in the population of size $N$. If the average fitness of the total population is $f_{av}$ the growth of membership of a Schema $H$ in successive generations is given by Eq. 2 as follows.

$$m(H, t+1) = m(H, t) * f(H) / f_{av} \tag{2}$$

Eq. 2 indicates that any Schema present in solutions with above-average fitness will grow over number of generations. It can be observed that a Schema with fitness as $c * f_{av}$ where $c$ is a constant, will grow exponentially, i.e. at a rate of $(1 + c)^t$.

*Crossover and mutation.* The solution being represented by an integer string, we may employ some suitable crossover selected from the order, partially mapped and cycle crossover operations. If the suitability of these crossover operations is to be quantified, we need to consider the probability of survival of an arbitrary Schema in the next generation after any of these crossover operation. A qualitative analysis of these three crossover operations in this regard is presented below.

Consider the example Schema given in Fig. 3. The order and the defining length of the Schema are indicated in the Fig. 3. To obtain an expression for the net survival factor for such Schema we need to consider the probability of survival of this Schema after the specific crossover operation. Let $P_1$ and $P_2$ be the probabilities that a Schema survives from the parent 1 and parent 2 respectively.

*Order crossover.* The probability that the given Schema survives from the parent 1 ($P_1$) is nearly equal to the probability that the cut occurred only from $l_r$ to $n - 1$. Hence

$$P_1 = (n - l_r)/(n - 1).$$

The probability that the given Schema survives from parent 2 ($P_2$) is nearly equal to the probability that no specified gene has occurred before the cut and all the gene string of the defining length has occurred in that order in the parent 2. Hence

$$P_2 \approx (1 - 1/n)^0 * (1/n)^{\delta}.$$

The net survival probability can be obtained from $P_1$ and $P_2$ as

$$\begin{aligned} \text{Net survival probability} &= P_1 + P_2 \\ &= (n - l_r)/(n - 1) \\ &\quad + (1 - 1/n)^0 * (1/n)^{\delta}. \end{aligned}$$

*Partially mapped crossover.* It can be observed that by this method of crossover many genes from parent 1 get preserved in the left cut segment, and many remaining genes from parent 2 get preserved in the right cut segment. Hence, the survival of a Schema in the offspring can be approximately

taken as its survival in the left, or right cut segments.

The probability of survival in the left cut segment $\approx l_l/(n-1)$

The probability of survival in the right cut segment $\approx (n-1-l_r)/(n-1)$.

From these two the net survival probability can be approximated as $(n-1-\delta)/(n-1)$.

*Cycle crossover.* By this method of crossover every gene is inherited in place from one of the parents. Let $p$ be the number of genes inherited from the parent 1. Hence $(n-p)$ genes are inherited from the parent 2. The net survival probability of the Schema is given by

Net survival probability $= (p/n)^0 * (1-p/n)^0$

In the approximate analysis given above, the survival of a Schema after the crossover operation is considered. If mutation operation is also taken into consideration the survival probability gets modified as follows. The Scheme gets disrupted by mutation only if a specified gene of the Schema is selected in course of mutation operation. The probability of such disruption is approximately given by $(1/n)^0$.

Hence, the survival probability after mutation is $(1-(1/n)^0)$. The overall survival factor associated with the Schema may be approximated as the sum of the survival probability after crossover and the survival probability after mutation. It is known that the rate of growth of better fit Schema depends on this survival factor.

A closer look is necessary at what type of Schema need to survive in the eventual construction of a near optimal solution for this embedding problem. A low order Schema means that we have only a few genes with specified values. Consider a cluster of $k+1$ consecutive nodes of the guest graph. If these are assigned with adjacent codes they occupy different gene positions separated by small values. This makes the Schema to have a small defining length. For example a Schema described by (* 0124***) of 8 genes has a defining length of 4. The Schema has a cluster of $\{V_{g2}, V_{g3}, V_{g4}, V_{g5}\}$ and these nodes are assigned with adjacent codes. Clusters of such nodes with adjacent codes, can be observed to correspond to low order small defining length Schema. Growth of such Schema would lead to some high order and long defining length Schema over some generations, ultimately leading to the optimal or near optimal solution. One may note that the order in which the codes are assigned to the nodes of the guest graph is not important. Only the adjacent codes given to the clusters of nodes is important. Hence, the order and partially mapped crossovers are more suitable for this problem than the cycle crossover. It may be noted that the cycle crossover preserves the inplace inheritance of genes, and depends only on the order of Schema. Comparatively the partially mapped crossover gives rise to a better survival probability for an arbitrary Schema. Let us consider an example Schema with $n=12$, $l_r=8$, and $l_l=4$ resulting in $o=2$ and $\delta=4$. The net survival probability after the order crossover may be observed to be 4/11 where as it is nearly 8/11 after the partially mapped crossover. This shows that partially mapped crossover performs better than the order crossover for this embedding problem. Experimentally the same has been observed. However, a good mix of some better fit low order and small defining length Schema in the initial population is essential. The above presented qualitative analysis does not prove that GA would lead to the optimal solution. It only reiterates the building block concept presented by John Holland. In this work, our aim is to study the efficacy of GA for embedding problem. This can be established only by experimentation. The details of the experimentation conducted is presented in the following section.

Table 1
MCNC Benchmark FSMs and their characteristics

| FSM name | States | Inputs | Outputs |
|---|---|---|---|
| bbra | 10 | 4 | 2 |
| bbsse | 16 | 7 | 7 |
| bbtas | 6 | 2 | 2 |
| cse | 16 | 7 | 7 |
| dk14 | 7 | 3 | 5 |
| dk16 | 27 | 2 | 3 |
| dk17 | 8 | 2 | 3 |
| dk27 | 8 | 1 | 2 |
| dk512 | 15 | 1 | 3 |
| donfile | 24 | 2 | 1 |
| ex1 | 20 | 9 | 19 |
| ex2 | 19 | 2 | 2 |
| ex3 | 10 | 2 | 2 |
| ex4 | 14 | 5 | 9 |
| ex5 | 9 | 2 | 2 |
| ex6 | 8 | 5 | 8 |
| lion9 | 9 | 2 | 1 |
| mark1 | 15 | 5 | 16 |
| planet | 48 | 7 | 19 |
| shiftreg | 8 | 1 | 1 |
| sla | 20 | 8 | 6 |
| train11 | 11 | 2 | 1 |

## 6. Experimental results

The problem of encoding FSM symbolic states is considered, for conducting the required experimentation. All the three solution methods are employed to get an embedding solution on a number of MCNC benchmark FSMs. The features of these FSMs such as the number of the states, primary inputs and outputs are listed in Table 1. In case of each FSM the encoding affinity between every pair of states is computed using the MUSTANG FANIN and FANOUT approaches. As the aim of the experimentation is to test the efficacy of GA for embedding, encoding affinity is computed by these simple methods. *It may be noted that which ever way the encoding affinity is computed the encoding procedure that determines the codes assigned to the symbolic states needs the solution for the embedding problem.*

The embedding solution is first obtained using the greedy heuristic method. This embedding may leave some codes unassigned to any state. These unassigned codes are considered for grouping with some assigned codes to form a seed group. The seed groups placed in a seed table help the generation of the initial population in the GA formulation. From the pool of the unassigned codes each code is grouped with an assigned code of the greedy solution, such that they are $k$-distant to each other. These seed groups are placed in the seed table. A random shuffling of one column of the seed table is employed for the generation of a random solution.

Since partially mapped crossover is shown to be more suitable for this problem the same is employed to obtain the embedding solution by the GA. A population of 40 candidate solutions, crossover rate of 25%, mutation of one gene or at most 2% of genes are employed as GA parameters. The GA required processing of at least 500 generations to reach the final solution. To understand how these parameters influence the quality of the GA solution, the performance of GA with variations in population size, crossover rate and mutation rate have been tested. A few test runs confirmed that the above choice of the GA parameters is adequate to give good performance by GA.

The embedding solution by SA for the same data sets of the benchmark FSMs are also obtained. In the SA formulation the starting temperature is chosen as $10^5$, and choice of smaller initial temperature resulted in poor final solutions. The number of perturbations per temperature are chosen as 20. It was observed that a choice of less number of perturbations degraded the SA performance, and a choice of more did not improve the performance. A value of 0.99 for $\alpha$ has been used for the temperature schedule. Any faster cooling resulted in poor

Table 2
Comparision of greedy, SA, and GA methods – Encoding affinity by MUSTANG-P

| Benchmark FSM | Greedy Cost | GA | | SA | |
|---|---|---|---|---|---|
| | | Cost | Time | Cost | Time |
| bbara | 530 | 520 | 0 m.25 sec. | 520 | 2 m.45 sec. |
| bbsse | 4976 | 4906 | 1 m.17 sec. | 4943 | 7 m.16 sec. |
| bbtas | 30 | 24 | 0 m.7 sec. | 25 | 0 m.48 sec. |
| cse | 9505 | 9332 | 1 m.21 sec. | 9343 | 6 m.56 sec. |
| dk14 | 1631 | 1547 | 0 m.10 sec. | 1547 | 1 m.6 sec. |
| dk16 | 3607 | 3473 | 5 m.13 sec. | 3347 | 23 m.38 sec. |
| dk17 | 330 | 316 | 0 m.14 sec. | 316 | 1 m.31 sec. |
| dk27 | 21 | 19 | 0 m.13 sec. | 21 | 1 m.27 sec. |
| dk512 | 143 | 131 | 1 m.6 sec. | 147 | 6 m.21 sec. |
| donfile | 11908 | 11876 | 3 m.53 sec. | 11888 | 19 m.15 sec. |
| ex1 | 33982 | 32246 | 2 m.25 sec. | 32254 | 12 m.50 sec. |
| ex2 | 12018 | 11796 | 2 m.2 sec. | 11840 | 11 m.16 sec. |
| ex3 | 2316 | 2246 | 0 m.24 sec. | 2246 | 2 m.38 sec. |
| ex4 | 124 | 120 | 0 m.55 sec. | 131 | 5 m.16 sec. |
| ex5 | 1582 | 1456 | 0 m.18 sec. | 1460 | 2 m.7 sec. |
| ex6 | 1136 | 1125 | 0 m.14 sec. | 1125 | 1 m.27 sec. |
| lion9 | 272 | 264 | 0 m.20 sec. | 266 | 2 m.7 sec. |
| mark1 | 4582 | 4532 | 1 m.6 sec. | 4528 | 6 m.3 sec. |
| planet | 79873 | 76292 | 25 m.16 sec. | 76292 | 2 h.13 m.53 sec. |
| shiftreg | 48 | 40 | 0 m.13 sec. | 40 | 1 m.27 sec. |
| sla | 4314 | 4132 | 2 m.29 sec. | 4142 | 12 m.14 sec. |
| train11 | 585 | 585 | 0 m.31 sec. | 589 | 3 m.12 sec. |

final solutions. It may be reiterated that these parameters decide the time efficiency of SA.

The cost of the solution by GA, SA, and the greedy methods in case of each MCNC benchmark FSM is presented in Tables 2 and 3. The time taken by GA as well as SA is presented in these tables. It may be observed that the greedy method gives a suboptimal solution in a very short time and hence, this time is not presented in the results of Tables 2 and 3.

The following may be observed from the results presented in the Tables 2 and 3. In case of each FSM, GA terminated with a solution of lower cost value than that by the greedy method. The cost of the final solution by GA is better than that by SA in case of many benchmark FSMs. However, for two instances SA final solutions are found to be better. The improvement obtained by GA over the greedy method is in the range of 1 to 10 percent in case of each of the FSM tested. The ratio of the time taken by SA to that by GA for each instance is nearly 6 to 1. These results experimentally establish the efficacy of GA for the embedding problem.

## 7. Conclusions

The formulation of the problem of embedding a complete graph in a minimum sized hypercube is presented. The practical importance of this

Table 3
Comparision of greedy, SA, and GA methods – Encloding affinity by MUSTANG-N

| Benchmark FSM | Greedy cost | GA | | SA | |
|---|---|---|---|---|---|
| | | Cost | Time | Cost | Time |
| bbara | 4258 | 4177 | 0 m.26 sec. | 4166 | 2 m.39 sec. |
| bbsse | 2262 | 2241 | 1 m.18 sec. | 2260 | 6 m.56 sec. |
| bbtas | 436 | 429 | 0 m.8 sec. | 429 | 0 m.58 sec. |
| cse | 8605 | 8443 | 1 m.20 sec. | 8477 | 7 m.2 sec. |
| dk14 | 3634 | 3472 | 0 m.10 sec. | 3472 | 1 m.8 sec. |
| dk16 | 14503 | 13371 | 5 m.14 sec. | 13706 | 22 m.59 sec. |
| dk17 | 773 | 773 | 0 m.12 sec. | 773 | 1 m.27 sec. |
| dk27 | 74 | 74 | 0 m.13 sec. | 75 | 1 m.28 sec. |
| dk512 | 416 | 402 | 1 m.9 sec. | 426 | 6 m.8 sec. |
| donfile | 11730 | 10144 | 3 m.51 sec. | 10942 | 18 m.3 sec. |
| ex1 | 33569 | 30685 | 2 m.22 sec. | 30962 | 12 m.13 sec. |
| ex2 | 4628 | 4406 | 2 m.24 sec. | 4463 | 11 m.17 sec. |
| ex3 | 1054 | 1030 | 0 m.25 sec. | 1040 | 2 m.39 sec. |
| ex4 | 29 | 27 | 0 m.50 sec. | 34 | 5 m.17 sec. |
| ex5 | 806 | 806 | 0 m.19 sec. | 806 | 2 m.7 sec. |
| ex6 | 704 | 652 | 0 m.13 sec. | 656 | 1 m.26 sec. |
| lion9 | 633 | 573 | 0 m.19 sec. | 591 | 2 m.8 sec. |
| mark1 | 587 | 550 | 1 m.7 sec. | 563 | 6 m.6 sec. |
| planet | 5090 | 5070 | 25 m.51 sec. | 5145 | 2 h.13 m.9 sec. |
| shiftreg | 88 | 88 | 0 m.13 sec. | 100 | 1 m.27 sec. |
| sla | 10730 | 10652 | 2 m.22 sec. | 10906 | 12 m.24 sec. |
| train11 | 590 | 550 | 0 m.34 sec. | 562 | 3 m.13 sec. |

problem is explained. The way to extend this formulation of embedding arbitrary graphs into hypercubes is pointed out. A comparative study of three different solution methods for this problem is conducted.

The different components of the SA, and GA solution methods, and their design aspects are explained. In case of GA formulation, three different crossover operations are compared. An approximate analysis of Schema survival after each of these crossovers, is presented. The specific embedding problem considered in this work is pointed out to be equivalent to the problem of encoding the symbolic states of a FSM. Greedy, SA, and GA solution methods for this application problem have been compared.

The experimental results show that the GA paradigm is more time efficient than SA for this embedding problem. It is also shown that using GA paradigm, better solutions (closer to optimal solution) than the solutions by greedy methods can be obtained. The improvement obtained by GA method when compared with the greedy method is about 1 to 10 percent in cases of various benchmark FSMs that have been tested. For large sized and more complex FSMs even, the time taken by the GA method is within reasonable limits. More importantly this time is observed to be much less

than the time taken by the SA method. The ratio of the times taken by SA and GA methods is between 5 and 6 in various cases of FSMs.

The parameter tuning of GA is a much simpler process than that of finding a suitable cooling schedule in case of SA. It may be pointed out that any well designed temperature schedule can reduce the SA time by some percentage. However, such design of temperature schedule is cumbersome and more time consuming. Choice of starting solution in case of SA also effects the end result of SA, but this is eliminated by forcing the starting solution as the greedy solution. To have the same advantage for GA, the greedy solution is chosen as one candidate solution in the starting pool of the solutions.

The experimental results reveal the efficacy of the GA paradigm for the important problem of embedding complete graph in a minimum sized hypercube.

## References


[1] D.B. Armstrong, A programmed algorithm for assigning internal codes to sequential machines, *IRE Trans. Electronic Comput.* EC-11 (Aug. 1962) 466–472.

[2] R.K. Brayton, G.D. Hachtel, C.T. Mcmullen and A. Sangiovanni Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer Academic, Hingham, MA, 1984).

[3] G. DeMicheli, R.K. Brayton and A. Sangiovanni Vincentelli, Optimal state assignment for finite state machines, *IEEE Trans. Comput. Aided Design* 4 (July 1985) 269–284.

[4] S. Devadas, H.K. Ma, A.R. Newton and A. Sangiovanni Vincentelli, MUSTANG: State asignment of finite state machines targeting multilevel logic implementation, *IEEE Trans. Comput. Aided. Design.* 7(12) (Dec. 1988) 1290–1299.

[5] Xuejun Du, G. Hachtel, B. Lin and A.R. Newton, MUSE: A multilevel symbolic encoding algorithm for state assignment, *IEEE Trans. Comput. Aided Design*, 10(1) (Jan. 1991) 28–38.

[6] M. Gary and D. Johnson, *A Guide to the Theory of NP-Completeness* (Freeman, San Fransisco, 1979).

[7] A. Gregori, Unit length embedding of binary trees on a square grid, *Informat. Process. Letters* 31(4) (May 1989) 167–173.

[8] J. Hartmanis, On the state assignment problem for sequential machines, *IRE Trans. Electronic Comput.* EC-10 (June 1961) 157–165.

[9] J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, MI, 1975).

[10] Young Man Kim and Ten-Hwang Lai, The complexity of congestion-1 embedding in a hypercube, *J. Algorithms* 12 (1991) 246–280.

[11] S. Kirkpatrick, M.P. Vecchi and C.D. Gellatt, Optimization by simulated annealing, *Science* 220 (May 1983) 671–680.

[12] D.W. Krumme, K.N. Venkataraman and G. Cybenco, *Hypercube Multiprocessor* (SIAM Publications, Philadelphia, 1986).

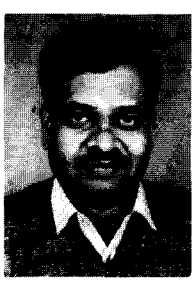
[13] Ten-Hwang Lai and A.P. Sprague, Placement of processors of a hypercube, *IEEE Trans. Comput.* 40(6) (June 1991) 714–722.

[14] S. Lakshmivarahan and S.K. Dhall, *Analysis and Design of Parallel Algorithms.* (McGraw-Hill, New York, 1990).

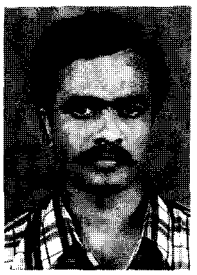
[15] B. Mitra and P. Palchoudhuri, A simulated annealing based state assignment approach for control synthesis. *Proc. 4th CSI/IEEE Int. Symp. on VLSI Design*, New Delhi, India (1991) 45–49.

[16] K. Shahookar and P. Majumder, A genetic approach to standard cell placement using meta-genetic parameter optimization, *IEEE Trans. Comput. Aided Design* 9(5) (May 1990) 500–512.

[17] Tiziano Villa and A. Sangiovanni Vincentelli, NOVA: State assignment of finite state machines for optimal two level logic implementation, *IEEE Trans. Comput. Aided Design* 9(9) (Sep. 1990) 905–924.

**R. Chandrasekharam** received B.E. and M. Tech degrees from Andhra and Kakatiya Universities of India respectively. He has been working for Ph.D in the Dept. of CSE, IIT, Kharagpur, India during 1990–93, and currently employed as Asst. Professor in the Dept. of CSE Regional Engineering College, Warangal, India. His interested areas of research are Optimization, Graph Algorithms, VLSI design, Neural Networks, Artificial Intelligence and Computer Communications.



**V.V. Vinod** has received his B.Tech (Hons), M. Tech and Ph.D degrees in Computer Science and Engineering from IIT, Kharagpur, India in 1988, 1990 and 1994 respectively. He has worked as a Junior Scientific Officer at IIT, Kharagpur from 1989 to 1993. From 1993 onwards he is working as a Member Technical Staff at Electronics Research and Development Centre, Trivandrum, India. His areas of interest include Neural Networks, Pattern Recognition, Machine Learning and Genetic Algorithms.

**S. Subramanian** has received M. Tech degree from PSG College of Technology Coimbatore, and Ph.D from IIT, Kharagpur. He has been working as a Professor in the Dept. of CSE IIT Kharagpur and presently holding the post of Principal, KCT Coimbatore India. His areas of research include Optimization, Neural Networks, Genetic Algorithms, Pattern Recognition, VLSI Design, Control Systems, Artificial Intelligence and Machine Learning.