



Meta-Heuristic Algorithms for FPGA Segmented Channel Routing Problems with Non-standard Cost Functions

SANCHO SALCEDO-SANZ*[†]

sancho.salcedo@uah.es

Departamento de Teoría de la Señal y Comunicaciones, Universidad de Alcalá, Spain

YONG XU

XIN YAO

School of Computer Science, The University of Birmingham, UK

Submitted April 26, 2004; Revised April 4, 2005

Published online: 12 August 2005

Abstract. In this paper we present three meta-heuristic approaches for FPGA segmented channel routing problems (FSCRPs) with a new cost function in which the cost of each assignment is not known in advance, and the cost of a solution only can be obtained from entire feasible assignments. Previous approaches to FSCRPs cannot be applied to this kind of cost functions, and meta-heuristics are a good option to tackle the problem. We present two hybrid algorithms which use a Hopfield neural network to solve the problem's constraints, mixed with a Genetic Algorithm (GA) and a Simulated Annealing (SA). The third approach is a GA which manages the problem's constraints with a penalty function. We provide a complete analysis of the three metaheuristics, by tested them in several FSCRPs instances, and comparing their performance and suitability to solve the FSCRPs.

Keywords: FPGAs, segmented channel architecture, hybrid algorithms, genetic algorithms, simulated annealing

1. Introduction

Field Programmable Gate Arrays (FPGAs) are a recently developed approach to the implementation of Application Specific Integrated Circuits (ASIC), which combine the flexibility of mask programmable gate arrays with the convenience of field programmability [6, 23]. FPGAs basically consist of regular arrays of routing networks, and logic cells, which can be programmed by users as logic modules in order to implement various types of logic functions [6].

Among the different types of existing FPGA architectures, one of the most studied is the row-based segmented channel routing model [4, 7, 8, 11, 16, 18, 19]. This architecture consists of rows of *logic cells* and *routing channels*. The interconnection between the logic cell pins and external I/O pins is performed by a combination of vertical and horizontal signal lines in the routing channel, and is called a *net* [6]. Each vertical line is called a *vertical segment*, however, no vertical constraint is considered in this model. Each horizontal line is laid out along *tracks* of the channel, and it is divided into horizontal segments. The

*Corresponding author.

[†]This work has been partially supported by a research project of the Universidad de Alcalá, project number UAH PI2005/078.

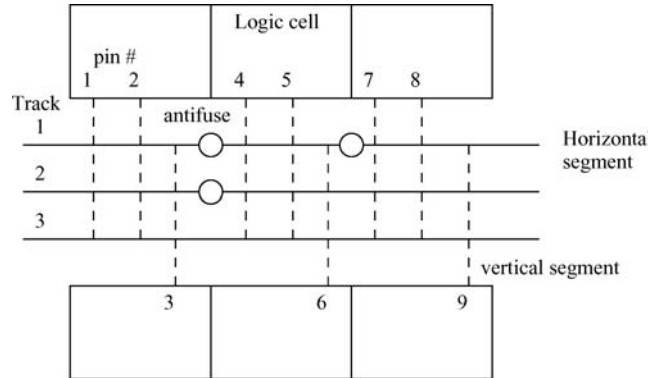


Figure 1. FPGA segmented channel routing architecture.

position and length of a horizontal segment is predefined, and may be different in every track. Nets can use the same track as long as no two nets share the same segment.

Programmable switches are located at each crossing point of vertical and horizontal segments, called *cross-fuses*, and others are located between two adjacent horizontal segments on the same track, called *antifuses*. Figure 1 shows an FPGA architecture formed by two rows of three logic cells, and one channel of three tracks. In this example, track 1 is composed of three horizontal segments separated by two antifuses. Imagine that two nets have to be routed, one connecting pins 1 and 3 and another one connecting pins 5 and 6. These two nets can be assigned to track 1 or 2 simultaneously, since they occupy different segments there. However, they cannot be assigned to track 3, because they would share segments in this track. If a net for pins 1 and 4 is assigned to track 1 or 2, the corresponding antifuse has to be programmed.

Various aspects of FPGAs have been previously studied. The FPGA segmented channel architecture was introduced by Gamal et al. in [7]. Green et al. formulated the FPGA segmented channel routing problem (FSCR) in [7, 8, 19] and proved that it is NP-complete. Further work about the NP-completeness of the FSCR was carried out by Li in [16]. Burman et al. presented and studied the staggered nonuniform length segmentation design problem for this architecture, in which a channel is divided into several regions, and each region consists of several tracks with equal length segments arranged in staggered fashion [4]. They also introduced a greedy algorithm in [4] and [23] called FSCR, which assigns each net to the track with the minimum cost among the available ones. This assignment is done sequentially, in descending order of nets length. Another greedy algorithm for the FSCR was proposed by Roy [18], which introduces a backtracking procedure in order to cope with the case of no feasible assignment of a net to a track. Another important approach for solving the FSCR is due to Funabiki et al. [6] who proposed a Gradual Hopfield neural network which was shown to improve previous approaches to the FSCR.

All the previous mentioned approaches to the FSCR deal with routing problems in which the cost of assigning a given net i to a track j (w_{ij} hereafter) is known a priori. That is, one can always calculate the cost associated with a given single assignment of a net to a track, without having an entire solution to the problem. All the greedy algorithms proposed for the FSCR are based on this observation. Also, the Gradual neural network approach in

[6] involves several heuristics to improve its convergence which are based on the previous knowledge of the cost associated with the assignment. However, there are situations in which the cost associated with one single assignment is unknown until an entire solution is obtained. Take as an example the case in which the cost function involves relations among assignments or depends on the number of nets assigned to a track. In general, if the FSCRП cost function is such that $f = f(\mathbf{X})$, where \mathbf{X} is an entire solution of the problem, the previous approaches to the FSCRП cannot be applied. We call these cost functions *non-standard* cost functions for the FSCRП.

In this paper we propose and analyze three meta-heuristic algorithms to cope with FSCRПs with non-standard cost functions $f = f(\mathbf{X})$. We propose two hybrid approaches which consist of a Hopfield Neural Network (HNN) for solving the FSCRП constraints, hybridized with two different global search heuristics for improving the quality of the solutions found by the HNN. The first global search heuristic we consider is a Genetic Algorithm (GA) [9]. The second one is a Simulated Annealing (SA) [15]. We also propose a GA with a penalty function for managing the problem's constraints. We will give the complete description of these three meta-heuristic, GAHNN, SAHNN, and GAPenalty, comparing their performance in several FSCRП test problems.

The rest of the paper is organized as follows: in the next section we describe the FSCRП, and the non-standard functions we use in our work. Section 3 describes the three meta-heuristic algorithms we propose. It is split in four subsections where we describe the HNN used, the GA and the SA algorithms used as global search heuristics in the hybrid approaches, and the GA with penalty function. In Section 4 we provide some simulations and results, comparing the performance of our proposed algorithms. Finally, Section 5 concludes the paper with some remarks.

2. Problem definition

In this section we provide the definition of the problem following the approach in [6], and present the new class of cost functions this paper deals with.

Consider a FPGA segmented channel routing architecture with M tracks, L columns, and a set of N nets. An antifuse is placed between two columns on a track to form segments. The location of antifuses is given by a matrix \mathbf{F} , in which an element f_{jk} , $j = 1, \dots, M$, and $k = 1, \dots, L$, is 1 if an antifuse is located between columns k and $k + 1$ in track j (see Figure 2(a)), or 0 otherwise. Net i , $i = 1, \dots, N$, is denoted by a pair of leftmost and rightmost columns as $(\text{left}_i, \text{right}_i)$. The FSCRП consists of assigning each net to one and only one track, in such a way that two nets cannot be assigned to the same track if they share segments. Mathematically, the FSCRП can be defined as follows: Find an assignment \mathbf{X} which minimizes

$$f = f(\mathbf{X}),$$

subject to:

$$\sum_{j=1}^M x_{ij} = 1, \tag{1}$$

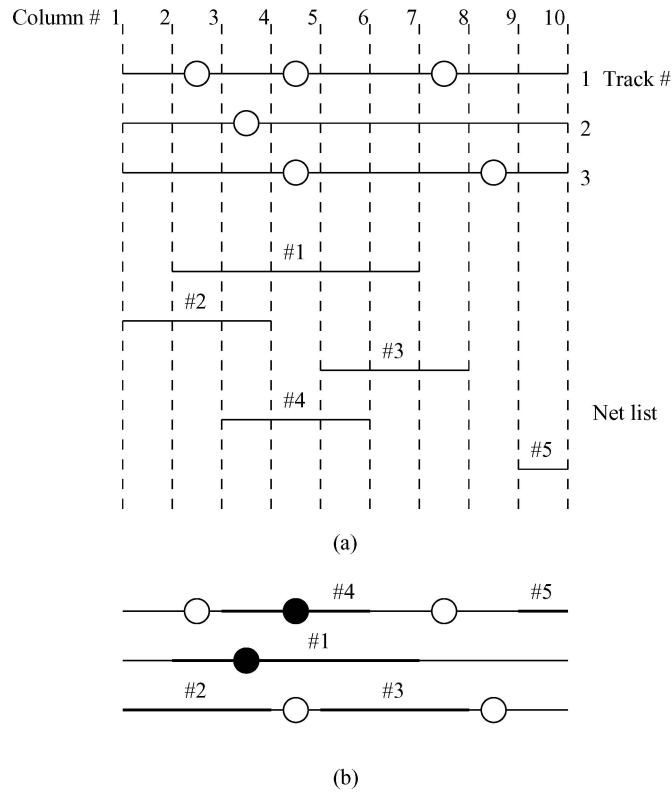


Figure 2. (a) FPGA segmented channel routing problem instance; (b) A feasible routing solution.

and

$$\sum_{j=1}^M \sum_{\substack{k=1 \\ k \neq i}}^N d_{jki} x_{ij} x_{kj} = 0, \quad \text{for } i = 1, \dots, N. \tag{2}$$

where d_{jki} is 1 if nets i and k are not allowed to share the same segment on track j , and 0 otherwise. Following [6], d_{jki} can be defined by

If $f_{jp} = 1$ for $p \in \{1, \dots, s, L\}$ with

$$\text{right}_i \leq p < \text{left}_k \text{ or } \text{right}_k \leq p < \text{left}_i, \tag{3}$$

then $d_{jki} = 0$, else $d_{jki} = 1$.

We now consider a small FSCR problem instance.

2.1. An example

Figure 2(a) shows the instance, formed by a channel of 3 tracks and 10 columns, in which 5 nets have to be assigned for interconnecting columns. Six antifuses are located in this channel, and the matrix \mathbf{F} of antifuses is given by $f_{12} = f_{14} = f_{17} = f_{23} = f_{34} = f_{38} = 1$, with the rest of the elements equal to 0. A feasible solution is depicted in Figure 2(b).

2.2. FSCRП objective functions

Several objective functions can be considered when solving an FSCRП instance, depending on the FPGA designer's needs. Roy [18] and Funabiki et al. [6] proposed a cost function f which only depends on the number of programmed antifuses as:

$$w_{ij} = \sum_{k=\text{left}_i}^{\text{right}_i-1} f_{ik}, \quad (4)$$

and the cost of a given solution \mathbf{X} would be

$$\sum_{i=1}^N \sum_{j=1}^M w_{ij} x_{ij}. \quad (5)$$

Another possible cost function is given by the maximum number of programmed antifuses per net. It is also possible to take into account the total occupied segment length as an objective function. All these objective functions are focused on reducing the delay associated with programming antifuses, and have been used before in the literature, so we call them *standard* objective measures for the FSCRП.

In this work we deal with a different set of objective functions in which the cost of a single assignment of a net to a track is unknown, until the complete solution is available. The importance of these functions is that they allow managing different priorities in the design of FPGAs, for example in situations in which, in addition to minimizing the number of programmed antifuses, the number of nets on each track have to be controlled. Other types of constraints involving relations between nets and tracks can be managed by using non-standard objective functions. Since, to our knowledge, there are no previous approaches managing this class of functions, we refer to them as *non-standard* objective functions for the FSCRП.

The most intuitive example of a non-standard objective function for the FSCRП consist in minimizing the number of antifuses programmed, having a balanced number of nets on each track.

$$f(\mathbf{X}) = a \cdot \sum_{j=1}^M \text{bal}_j + b \cdot \sum_{i=1}^N \sum_{j=1}^M w_{ij} x_{ij}. \quad (6)$$

where $\sum_{j=1}^M \text{bal}_j$ ensures that the assignment of nets to tracks is balanced over all the tracks. The term bal_j can be defined as:

$$\text{bal}_j = \begin{cases} 10, & \text{if } \sum_{i=1}^N x_{ij} = N_T, \\ 20 \cdot \text{abs} \left(N_T - \sum_{i=1}^N x_{ij} \right), & \text{otherwise.} \end{cases} \quad (7)$$

where N_T is the required number of nets per track. Note that using this function, the cost of assigning one single net to a track cannot be calculated, since we have to have the entire solution to calculate the term $\sum_{j=1}^M \text{bal}_j$. Note also that many other non-standard functions can be defined as cost functions for the FSCR, depending on the FPGA designer necessities. Finally, since the use of non-standard cost functions reduces the amount of information available in advance, it seems reasonable to use "blind" search heuristics such as genetic algorithms or simulated annealing in order to solve the FSCR.

3. Proposed approaches

In this section we describe the meta-heuristic approaches for the FSCR with non-standard functions we propose. First we describe the hybrid approaches, presenting the HNN used and the global search heuristics (a GA and a SA). After this, we describe the GA with penalty function.

3.1. Hybrid approaches for the FSCR

3.1.1. The Hopfield neural network. The Hopfield network we use as a local search algorithm for solving the FSCR constraints belongs to a class of binary Hopfield networks [24] where the neurons can only take values 1 or 0. The dynamics of this network depends on a matrix C which defines the minimum distance between two 1s in the network for each row, and on the initial state of the neurons. See [24] for further details. The structure of the HNN can be represented by a graph, where the set of vertices are the neurons, and the set of edges define the connections between the neurons. We map a neuron to every element in the solution matrix \mathbf{X} . In order to simplify the notation, we shall also use matrix \mathbf{X} to denote the neurons in the Hopfield network. The HNN dynamics can then be described in the following way: After a random initialization of every neuron with binary values, the HNN operates in a serial mode. This means that only one neuron is updated at a time, while the rest remain unchanged. Denoting by $x_{ij}(t)$ the state of a neuron at time t , the updating rule is described by:

$$x_{ij}(t) = \text{isgn} \left(\sum_{\substack{p=1 \\ p \neq i}}^N \sum_{\substack{q=\max(1, j-c_{j,i,p}+1) \\ q \neq j}}^{\min(M, j+c_{j,i,p})} x_{pq} \right) \forall i, j. \quad (8)$$

where the isgn operator is defined by:

$$\text{isgn}(a) = \begin{cases} 0 & \text{if } a > 0, \\ 1 & \text{otherwise} \end{cases}$$

Note that the updating rule only takes into account neurons x_{pq} with value 1 within a distance of c_{jip} . The matrix C is an $M \times N \times N$ matrix which encodes the problem's constraint given in Eqs. (1) and (2). Its elements depend on the matrix of fuses \mathbf{F} through the parameters d_{jki} defined in Section 2:

$$c_{jki} = \begin{cases} M, & \text{if } i = k, \forall j \\ d_{jki}, & \text{if } i \neq k, \forall j. \end{cases} \quad (9)$$

Note that for each row in \mathbf{X} , there is a $N \times N$ binary matrix which forces one and only one 1. However, there may be several 1s in the same column.

In the updating rule defined above, the neurons x_{ij} are updated in their natural order, i.e., $i = 1, 2, \dots, N, j = 1, 2, \dots, M$. We introduce a modification of this rule by performing the updating of the neurons in a random ordering of the rows (variable i). This way the variability of the feasible solutions found will increase. Let $\pi(i)$ be a random permutation of $i = 1, 2, \dots, N$. The new updating rule of the HNN is:

$$x_{\pi(i)j}(t) = \text{isgn} \left(\sum_{\substack{p=1 \\ p \neq \pi(i)}}^N \sum_{\substack{q=\max(1, j-c_{j,\pi(i),p}+1) \\ q \neq j}}^{\min(M, j+c_{j,\pi(i),p})} x_{pq} \right) \forall i, j. \quad (10)$$

The resulting updating rule runs over the rows of \mathbf{X} in the order given by the permutation $\pi(i)$, but the columns are updated in natural order $j = 1, 2, \dots, M$.

A *cycle* is defined as the set of $N \times M$ successive neuron updates in a given order. In a cycle, every neuron is updated once following the given order $\pi(i)$, which is fixed during the execution of the algorithm. After every cycle, the convergence of the HNN is checked. The HNN is considered converged if none of the neurons have changed their state during the cycle. The final state of the HNN dynamics is a potential solution for the FSCRCP, which fulfils the problem's constraints given in Section 2. Note, however, that the solution found may be unfeasible if some net is not assigned.

3.1.2. Hybrid approach GAHNN: The genetic algorithm. The GA we propose to be hybridized with the Hopfield network codifies a population of χ potential solutions for the FSCRCP as binary strings of length $N \times M$. Each string represents a different matrix \mathbf{X} . The population is then evolved through successive generations by means of the application of the genetic operators selection, crossover and mutation [9].

Selection is the process by which individuals in the population are randomly sampled with probabilities proportional to their fitness values. We consider the selection mechanism known as *roulette wheel*, in which the probability of a given individual to survive to the next generation is directly proportional to its associated fitness as:

$$P(I_j) = \frac{f(I_j)}{\sum_{i=1}^{\chi} f(I_i)} \quad (11)$$

where I_i represents an individual of the GA and $f(\cdot)$ is the associated fitness.

An elitist strategy, consisting of passing the individual with the highest fitness to the next generation, is used in our algorithm to preserve the best solution found so far in the

evolution. The set of individuals chosen in the selection procedure, of the same size as the initial population, is subjected to the crossover operation. First, the binary strings are coupled at random. Second, for each pair of strings, an integer position along the string is selected uniformly at random. Two new strings are composed by swapping the bits between the selected position and the end of the string. This operation is applied to the pair with probability P_c . After the crossover operator, each individual in the GA population is considered for mutation, with a probability P_m . Mutation consists of flipping the value of N_f random chosen bits in the selected individual from 1 to 0 or vice versa.

Other previous approaches to combinatorial optimization problems using a hybrid scheme Hopfield Network-Genetic Algorithm are the works by Watanabe et al. [27], Balicki et al. [2], Bousño-Calzón et al. [3] and Salcedo-Sanz et al. [21].

The complete algorithm for the FSCPR, formed by the GA and the HNN described in Section 3.1.1, is summarized below:

```

GAHNN algorithm
Initialize GA population at random
while(max. number of generations has not been reached) do
  for(every individual  $\mathbf{X}$ )
    Run the HNN to obtain a feasible  $\mathbf{X}$ .
    Calculate the fitness value of the individual  $f(\mathbf{X})$ .
    if  $\mathbf{X}$  is not feasible, apply a penalty to  $f(\mathbf{X})$ .
    Substitute the GA individual by the new  $\mathbf{X}$  obtained through the HNN.
  endfor
  selection
  crossover
  mutation
end(while)

```

3.1.3. Hybrid approach SAHNN: The simulated annealing. SA has been widely applied to solve combinatorial optimization problems [10, 14, 15, 26]. It is inspired by the physical process of heating a substance and then cooling it slowly, until a strong crystalline structure is obtained. This process is simulated by lowering an initial temperature by slow stages until the system reaches an equilibrium point, and no more changes occur. Each stage of the process consists of changing the configuration several times, until a thermal equilibrium is reached, and then a new stage starts, with a lower temperature. The solution of the problem is the configuration obtained in the last stage. In the standard SA, the changes in the configuration are performed in the following way: A new configuration is built by a random displacement of the current one. If the new configuration is better, then it replaces the current one, and if not, it may replace the current one probabilistically. This probability of replacement is high in the beginning of the algorithm, and decreases in every stage. The solution found by SA can be considered a "good enough" solution, but it is not guaranteed to be the best.

In this paper we consider the hybridization of a SA and the Hopfield neural network presented in Section 3.1.1 for solving the FSCR. The main idea behind this is that configurations involved in the SA are feasible solutions for the FSCR. The SA will then seek

the best feasible solution with respect to a given cost function, in this case a non-standard cost function for the FSCRCP.

There have been similar previous approaches to other optimization problems using a hybrid model Hopfield Network-Simulated Annealing as the works by Macías et al. [5] and Salcedo-Sanz et al. [22], SA hybridized with other optimization procedures as the works by Kim et al. [13] and Yao [28].

The most important parts in a SA algorithm are: the objective function to be minimized during the process, the chosen representation for solutions and the mutation or configuration change operator. The objective function to be minimized is the non-standard cost function for the FSCRCP defined in Section 2.2. The representation of the problem is the assignment matrix \mathbf{X} and the change from one configuration to another is performed by means of the standard flip mutation of a given number N_f of bits in \mathbf{X} .

The complete algorithm for the FSCRCP, formed by the SA and the HNN described in Section 3.1.1, performs in the following way:

```

SAHNN algorithm
k = 0;
T = T0;
Initialize a potential solution at random;
Run the HNN to obtain  $\mathbf{X}$ ;
evaluate( $\mathbf{X}$ , f( $\mathbf{X}$ ));
if  $\mathbf{X}$  is not feasible, apply a penalty to f( $\mathbf{X}$ ).
repeat
for j = 0 to  $\xi$ 
 $\mathbf{X}_{mut}$  = mutate( $\mathbf{X}$ );
Run the HNN to obtain  $\mathbf{X}$ ;
evaluate( $\mathbf{X}_{mut}$ , f( $\mathbf{X}$ ));
if  $\mathbf{X}$  is not feasible, apply a penalty to f( $\mathbf{X}$ ).
if((f( $\mathbf{X}_{mut}$ ) < f( $\mathbf{X}$ )) OR (random(0, 1) < e( $\frac{-\alpha}{T}$ ))) then
 $\mathbf{X}$  =  $\mathbf{X}_{mut}$ ;
endif
endfor
T = fT(T0, k);
k = k + 1;
until(T < Tmin);
    
```

In SAHNN, k counts the number of iterations performed; T keeps the current temperature; T_0 is the initial temperature; T_{min} is the minimum temperature to be reached; \mathbf{X} stands for the current configuration and \mathbf{X}_{mut} for the new configuration after the mutation operator is applied; f represents the cost function considered (see Section 2.2); ξ is the number of changes performed for a given temperature T ; f_T is the freezer function; and α is a constant. Parameter α and the initial temperature T_0 are chosen to have an initial acceptance probability about 0.8, a value usually used. The freezer function is defined as

$$f_T = \frac{T_0}{1 + k}. \quad (12)$$

The minimum temperature T_{\min} is calculated on the basis of the desired number of iterations as:

$$T_{\min} = f_T(T_0, \text{numIt}). \quad (13)$$

3.2. A GA with penalty function for the FSCRCP

Finally, we propose another approach for the FSCRCP, based on a GA with a penalty function for managing the problem's constraints. Instead of a binary representation, in this GA each solution \mathbf{X} is encoded as an integer string of length N , \tilde{x} , such that $\tilde{x}_i = j$ means that net i , $i = 1, \dots, N$ has been assigned to track j , $j = 1, \dots, M$. Note that, using this encoding method, there will be no infeasible solutions due to unassigned nets to tracks, but only due to unfeasible assignments.

The management of the problem's constraints is carried out by means of a term of penalty for solutions with infeasible assignments. We define the penalty function as the sum of two parts: First, a constant term is added to the fitness function in such a way that the fitness value of the best infeasible solution is greater than the fitness value of the worst feasible solution. The second term of the penalty is defined to be proportional to the number of infeasible assignments.

$$\text{Penalty} = c \cdot \text{numInf} \quad (14)$$

where numInf stands for the number of infeasible assignments and c is the penalty for one infeasible assignment, usually it is a parameter to be tuned for each problem.

This way of managing the problem's constraints is well known for the GA research community, and has provided good results in different problems of combinatorial optimization with constraints [12, 17].

The Selection and Crossover operators in this GA are the same as in our hybrid GAHNN approach (see Section 3.1.2). The mutation operator consists of changing the value of N_f randomly chosen genes by a different value in $\{1, \dots, M\}$. The probability of mutating one individual is P_m , equal to the probability of mutation in the GAHNN algorithm.

The complete algorithm for the FSCRCP, can be summarized in the following way:

GApenalty algorithm

```

Initialize GA population at random
while(max. number of generations has not been reached) do
  for(every individual  $\tilde{x}$ )
    Calculate the fitness value of the individual  $f^* = f(\tilde{x}) + c \cdot \text{numInf}$ .
  endfor
  selection
  crossover
  mutation
end(while)

```

Table 1. Main characteristics of the instances tackled.

| Problem | Nets | Tracks | Columns |
|---------|------|--------|---------|
| 1 | 32 | 8 | 64 |
| 2 | 32 | 8 | 64 |
| 3 | 32 | 8 | 64 |
| 4 | 32 | 8 | 64 |
| 5 | 48 | 12 | 96 |
| 6 | 48 | 12 | 96 |
| 7 | 48 | 12 | 96 |
| 8 | 48 | 12 | 96 |
| 9 | 64 | 16 | 128 |
| 10 | 64 | 16 | 128 |
| 11 | 64 | 16 | 128 |
| 12 | 64 | 16 | 128 |
| 13 | 96 | 24 | 200 |
| 14 | 96 | 24 | 200 |
| 15 | 96 | 24 | 200 |
| 16 | 96 | 24 | 200 |
| 17 | 128 | 32 | 288 |
| 18 | 128 | 32 | 288 |
| 19 | 128 | 32 | 288 |
| 20 | 128 | 32 | 288 |
| 21 | 256 | 64 | 320 |
| 22 | 256 | 64 | 320 |
| 23 | 256 | 64 | 320 |
| 24 | 256 | 64 | 320 |

4. Numerical examples and performance analysis

4.1. Generation of the test instances

In order to test the performance of our approaches to the FSCRCP, twelve test instances of different length have been randomly generated. Table 1 shows the main characteristics of the instances tackled. There are 24 problems with different values for the number of nets, tracks and columns. The value of N_T is equal to 4 for all the instances, so in the non-standard function we consider each track must have 4 nets assigned. The nets were generated for having lengths between 1 and 8, by means of randomly generation the values $right_i$ and $left_i$ for each net i . The matrix of fuses \mathbf{F} was obtained by means of randomly assigning each element $f_{ij} = 1$ with a probability of 0.7 or $f_{ij} = 0$ with a probability of 0.3. If two 1s appeared together in the matrix, one of them was flipped into a 0. Also all the elements of the last column of matrix \mathbf{F} , f_{iL} , were fixed to 0.

| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|
| 1 | | ■ | | | | | ■ | | | | | | | ■ | | | | | | | | | | | | | ■ | | | | | | |
| 2 | | | | ■ | | | | | | | | | | | | | ■ | | | | | ■ | | | | | ■ | | | | | | |
| 3 | | | | | | ■ | | | | | | | ■ | | | | | | | ■ | | ■ | | | | | | | | | | | |
| 4 | | | | | | | | ■ | | ■ | | ■ | | | | | | | | | ■ | | ■ | | | | | | | | | | |
| 5 | ■ | | | | | | | | | | | | | | | ■ | | | | ■ | | ■ | | | | | | | | | ■ | | |
| 6 | | | ■ | | | | | | ■ | ■ | | | | | | | | | | | | | | | | ■ | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | ■ | | | | | | | | | | ■ | ■ | ■ | | | ■ |
| 8 | | | | | ■ | | | | | | | | | | ■ | | | | | | | | | | ■ | | | | | | | | ■ |

Figure 3. FPGA segmented channel routing problem solution found by the GAHNN algorithm in Problem 1.

4.2. Results

We run each algorithm 30 times for each problem, keeping the best, average and standard deviation values provided by each algorithm. The parameters of the non-standard cost function considered are $a = 0.6$ and $b = 0.4$ (see Section 2.2). Using these values we give slightly more importance to the balance of nets in the channel than to the number of programmed antifuses. Other values of a and b are possible depending on the FPGA designer necessities.

The GAHNN and GA with penalty function were run with a population of $\chi = 50$ individuals, 300 generations. The probability of crossover and mutation were set to $P_c = 0.6$ and $P_m = 0.01$. In the SAHNN algorithm, the parameter ξ (maximum number of mutations within a given temperature) was fixed to 50, with a maximum number of iterations $numIt = 300$. This way all the meta-heuristics tested performed the same number of function evaluations.

Table 2 shows the results obtained by the GAHNN, SAHNN and the GA with penalty function approaches. Table 3 shows the results of a t -test performed over the data obtained by the three compared algorithms. We perform this analysis in order to know if the differences in performance between algorithms are statistically significant or not. In the smallest Problems, 1 to 4, both hybrid approaches seem to perform better than the GA with penalty function, obtaining better results in best and average values in the 30 runs. The t -test values shown in Table 3 confirm this best performance of the hybrid approaches over the GA with penalty function. Note that the hybrid GAHNN performs statistically better than the SAHNN in Problems 1 and 4, whilst the performance of both algorithm is statistically equal for Problems 2 and 3. Figure 3 shows the best solution obtained by the GAHNN algorithm in Problem 1. Note that every track hold four nets, as was requested in the cost function by setting $N_T = 4$.

In problems 5 to 8, the approach GAHNN performs clearly better than the SAHNN and GA with penalty function. It is easy to see that the SAHNN degrades its performance in this second set of test instances. In fact, the GA with penalty function performs better than SAHNN in Problems 5 and 7, whereas there were no difference between both algorithms performance in Problems 6 and 8.

In Problems 9 to 12 the SAHNN algorithm performs significantly worse than the GAHNN and the GA with penalty function. GAHNN seems to have a better performance on these

Table 2. Comparison of the results obtained by the different algorithms considered. Best, Average and Standard Deviation values are provided.

| Problem | GAHNN | | | SAHNN | | | GAPenalty | | |
|---------|-------|-------|------|-------|-------|------|-----------|--------|------|
| | Best | Avg. | Dev. | Best | Avg. | Dev. | Best | Avg. | Dev. |
| 1 | 62.4 | 64.5 | 1.0 | 63.6 | 65.1 | 1.0 | 64.0 | 69.7 | 5.6 |
| 2 | 66.4 | 68.7 | 1.0 | 67.2 | 69.0 | 0.8 | 68.8 | 75.2 | 6.0 |
| 3 | 64.8 | 66.9 | 2.3 | 65.2 | 66.7 | 0.8 | 67.6 | 75.6 | 5.9 |
| 4 | 68.8 | 70.4 | 0.9 | 69.6 | 71.0 | 0.9 | 70.8 | 79.2 | 6.0 |
| 5 | 101.2 | 107.5 | 5.2 | 103.6 | 116.5 | 5.1 | 102.0 | 114.7 | 7.9 |
| 6 | 102.0 | 110.7 | 5.6 | 105.2 | 119.6 | 5.5 | 105.2 | 115.3 | 7.2 |
| 7 | 102.8 | 111.4 | 6.3 | 116.4 | 121.6 | 4.6 | 107.6 | 119.6 | 7.1 |
| 8 | 99.6 | 106.6 | 5.5 | 100.4 | 117.3 | 6.8 | 99.6 | 111.3 | 6.5 |
| 9 | 138.4 | 156.3 | 8.2 | 161.6 | 173.6 | 7.6 | 139.2 | 156.5 | 7.5 |
| 10 | 136.8 | 151.4 | 7.3 | 160.7 | 176.6 | 8.3 | 137.2 | 155.5 | 10.1 |
| 11 | 140.0 | 159.3 | 9.9 | 162.4 | 179.3 | 9.7 | 149.6 | 160.5 | 7.1 |
| 12 | 138.0 | 156.7 | 9.9 | 164.8 | 180.0 | 10.4 | 138.4 | 158.1 | 10.8 |
| 13 | 162.2 | 172.7 | 7.9 | 191.3 | 204.4 | 8.7 | 166.4 | 183.6 | 7.8 |
| 14 | 170.1 | 180.2 | 5.2 | 194.9 | 208.2 | 8.1 | 172.8 | 180.6 | 6.0 |
| 15 | 162.4 | 178.6 | 8.0 | 190.8 | 206.6 | 8.4 | 165.5 | 179.6 | 7.5 |
| 16 | 164.1 | 177.4 | 7.0 | 195.8 | 208.3 | 9.0 | 169.2 | 183.3 | 7.0 |
| 17 | 228.0 | 253.3 | 12.1 | 278.4 | 294.4 | 12.3 | 230.1 | 252.0 | 9.6 |
| 18 | 225.2 | 253.0 | 14.9 | 283.2 | 298.5 | 8.3 | 232.3 | 251.9 | 15.6 |
| 19 | 226.7 | 249.3 | 14.8 | 289.6 | 303.4 | 11.5 | 224.5 | 249.4 | 13.7 |
| 20 | 227.7 | 255.6 | 15.1 | 286.6 | 301.2 | 11.2 | 229.1 | 252.1 | 13.0 |
| 21 | 533.2 | 590.9 | 26.8 | 642.6 | 689.0 | 21.2 | 553.2 | 596.25 | 23.3 |
| 22 | 544.0 | 594.4 | 26.0 | 662.1 | 692.4 | 16.2 | 544.0 | 596.2 | 28.5 |
| 23 | 538.1 | 593.1 | 25.8 | 659.4 | 691.7 | 17.6 | 541.2 | 592.1 | 27.5 |
| 24 | 540.2 | 598.4 | 27.6 | 649.8 | 694.6 | 19.1 | 546.9 | 593.9 | 26.0 |

problems, but it is not statistically significant. However, the algorithm GAHNN obtained a better solution than GA with penalty function in all these set of problems.

This trend is maintained in the hardest instances, from 13 to 24. Note that only in Problem 13 and in 16 de GAHNN approach performs statistically better than the GA with penalty function. However, the GAHNN approach is able to obtain the best solution in the majority of the instances. The performance of the SAHNN approach is much worse than the other two algorithms analyzed in these instances.

4.3. Computational cost and convergence of the algorithms

One important point to be considered in the study of the algorithms presented is the real computational time consumed by them. It is expected that the hybrid approaches to be more

Table 3. t values obtained by a two-tailed t -test for Problems 1 to 12. † stands for values of t with 29 degrees of freedom which are significant at $\alpha = 0.05$.

| Problem | GAHNN-SAHNN | GAHNN-GApentalty | GApentalty-SAHNN |
|---------|-------------|------------------|------------------|
| 1 | -2.58† | -5.2† | 4.4† |
| 2 | -1.57 | -5.5† | 5.3† |
| 3 | 0.36 | -7.5† | 7.6† |
| 4 | -2.6† | -8.1† | 7.3† |
| 5 | -6.9† | -3.5† | -0.9 |
| 6 | -6.4† | -2.7† | -2.7† |
| 7 | -6.4† | -4.2† | -1.3 |
| 8 | -6.5† | -3.7† | -3.7† |
| 9 | -8.0† | -0.1 | -9.7† |
| 10 | -11.8† | -1.8 | -10.8† |
| 11 | -7.0† | -0.45 | -9.6† |
| 12 | -8.5† | -0.5 | -7.3† |
| 13 | -12.4† | -3.3† | -6.0† |
| 14 | -13.0† | -0.2 | -9.6† |
| 15 | -8.2† | -0.3 | -10.5† |
| 16 | -10.8† | -2.4† | -9.2† |
| 17 | -9.5† | -0.4 | -11.5† |
| 18 | -8.8† | -0.2 | -12.8† |
| 19 | -11.1† | -0.2 | -14.2† |
| 20 | -10.8† | 0.3 | -13.9† |
| 21 | -14.2† | -0.6 | -16.6† |
| 22 | -10.2† | -0.2 | -10.9† |
| 23 | -10.3† | 0.2 | -12.8† |
| 24 | -11.7† | 0.3 | -14.5† |

time consuming than the GA with penalty function, because the HNN convergence is a time consuming process, above all in large instances. In order to give an idea of running times, we have measured the real computational time¹ that the different algorithms needs to complete 15000 function evaluations. Note that the computation times of algorithms strongly depends on the simulation platform used for running the experiments. In our case it was a *Dual Xeon/2.8 GHz*. Table 4 displays the real computational time of the different algorithms considered. These results show that the hybrid approaches are more time consuming approaches than the GA with penalty function. The advantages of GApentalty over the GAHNN and SAHNN in computational time are easy to see, and they are more pronounced in large size problems, but, on the other hand, the GAHNN obtains solutions of better quality. Note that the SAHNN algorithm does not perform better than the GApentalty in medium and large size problems, and it is computationally more expensive.

Another interesting analysis is the HNN percentage of convergence, i.e., the probability that the HNN provides a feasible solution for the FSCR. Figure 4 shows the HNN

Table 4. Real computational time in seconds for the algorithms considered.

| Problem | GAHNN | SAHNN | GApentalty |
|---------|-------|-------|------------|
| 1-4 | 15 | 15 | 2 |
| 5-8 | 60 | 60 | 5 |
| 9-12 | 100 | 100 | 15 |
| 13-16 | 150 | 150 | 25 |
| 17-20 | 200 | 200 | 40 |
| 21-24 | 300 | 300 | 60 |

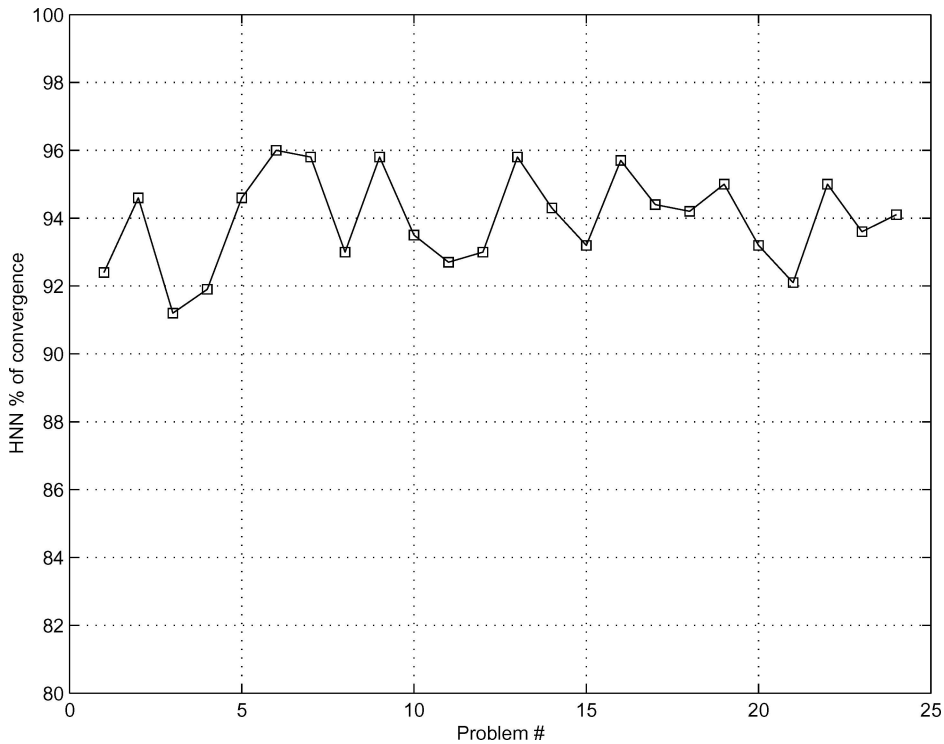


Figure 4. HNN percentage of convergence to feasible solutions in the test instances tackled.

percentage of convergence to feasible solutions for each problem considered. This calculation is based on the results obtained launching 1000 HNNs with random initial state and computing the number of unfeasible solutions obtained. Note that over 90 % of the solutions found by the binary HNN are feasible in all the test problems.

4.4. A lower bound for the non-standard cost function considered

In the experiments performed in this paper we have used the following non-standard cost function, which takes into account the number of antifuses programmed as well as the balance of nets into tracks:

$$f(\mathbf{X}) = 0.6 \cdot \sum_{j=1}^M \text{bal}_j + 0.4 \cdot \sum_{i=1}^N \sum_{j=1}^M w_{ij} x_{ij}. \quad (15)$$

where the term bal_j is defined as:

$$\text{bal}_j = \begin{cases} 10, & \text{if } \sum_{i=1}^N x_{ij} = N_T, \\ 20 \cdot \text{abs} \left(N_T - \sum_{i=1}^N x_{ij} \right), & \text{otherwise.} \end{cases} \quad (16)$$

Note that this is a non-standard cost function for the FSCRCP because it depends on the full solution to the problem X .

One approach often used in the literature is to define a lower bound (LB) for the objective function, and then to compare what is the gap between the LB and the best solution found by the algorithm. It is possible to define a LB for the FSCRCP objective function presented in this paper, in the following way:

Note that the objective function has two parts, one for balance and one for the cost of programming the antifuses. A LB for the balance part can be obtained as

$$10 \cdot M \leq \sum_{j=1}^M \text{bal}_j. \quad (17)$$

A LB for the cost of programming the antifuses can be obtained in the following way: First, let us consider a FSCRCP where we eliminate the constraint given by Eq. (2). In this framework, assign each net j to the truck in which less antifuses has to be programmed. Note that this provides a LB for the second term of the objective function.

As an example, consider the small problem in Section 2.1. It is formed by 5 nets, 3 trucks and 10 columns. Thus the LB for the balance term of the objective function is $10 \cdot 3 = 30$. The LB for the second term depends on each net: without considering constraint (2), if we assign net #1 to track 2 only 1 antifuse has to be programmed, assigning net #2 to track 3 no antifuse has to be programmed, and neither if we assign net #3 to tracks 2 or 3. Finally, 1 antifuse has to be programmed if we assign net #4 (tracks 2 or 3) and no antifuse is needed if net #5 is assigned to tracks 1, 2 or 3. It is straight forward to calculate that the LB for this term of the cost function is 2. Thus the final LB for the cost function is $0.6 \cdot 30 + 0.4 \cdot 2 = 18.8$. The best solution to this problem is given in Figure 5. If we calculate the real cost value for this solution (considering $N_T = 2$), it is $0.6 \cdot (10 + 10 + 20) + 0.4 \cdot 2 = 24.8$.

Table 5 shows the LB obtained for all the instances considered. If we compare this LB with the best solutions obtained by the algorithms, mainly the GAHNN and the GA with

Table 5. Lower Bound for the instances considered.

| Problem | Lower Bound |
|---------|-------------|
| 1 | 55.4 |
| 2 | 58.4 |
| 3 | 56.0 |
| 4 | 60.4 |
| 5 | 84.0 |
| 6 | 86.4 |
| 7 | 88.8 |
| 8 | 84.6 |
| 9 | 112.0 |
| 10 | 116.2 |
| 11 | 115.0 |
| 12 | 115.2 |
| 13 | 153.6 |
| 14 | 155.8 |
| 15 | 158.2 |
| 16 | 154.6 |
| 17 | 204.8 |
| 18 | 209.4 |
| 19 | 200.4 |
| 20 | 210.2 |
| 21 | 415.6 |
| 22 | 412.0 |
| 23 | 413.2 |
| 24 | 408.4 |

penalty function, it is possible to see that the gap between them is small for all instances. Note that the contribution of the balance term to the LB is larger than the contribution of the second term. Thus, this small gap between LB and the best solutions of the algorithms is an indication that the balance of nets into tracks is achieved by the algorithms in all the test instances.

4.5. Further analysis

In the previous sections, we have shown that our GAHNN approach is a very good option for solving the FSCRП with non standard cost functions. It could be also applied to solve FSCRП with standard functions, in which the cost of assigning a net to a track is known in advance. In this kind of situations, our hybrid metaheuristics does not perform better than greedy algorithms, existing in the literature. In order to show the differences in performance

we have implemented the greedy approach proposed in [1], adapting it to the FSCR. This algorithm is, in pseudo-code, as follows:

Pseudo-code of the Greedy algorithm in [1].

```

Choose a permutation  $\pi(l_N)$  of nets, at random.
for(each net  $\pi(l_i)$ )
  Determine  $\min(cost_{ij})$ , i.e. obtain the minimum cost of
  assigning net  $\pi(l_i)$  to a track  $j$ .
  Assign  $\pi(l_i)$  to  $j$  if it is available (Equation (2) is fulfilled).
  If this assignment is not possible, assign net  $\pi(l_i)$ 
  to the following available track with lower cost.
  Repeat the process until all the nets are assigned.
end(for)

```

Note that the number of function evaluations can be controlled by adjusting the number of permutations to be considered. Note also that this greedy algorithm may provide unfeasible solutions, if any of the nets is not assigned to a track.

Table 6 shows a comparison of the results obtained by the greedy algorithm and the GAHNN for the 12 first problems proposed. For these experiments, the number of function evaluations were fixed to 15000 for both algorithms. As standard cost function, we have used Eq. (5), in Section 2.2. Best solution obtained by the algorithms in 30 runs has been computed. It is easy to see that in FSCRPs with standard cost functions the GAHNN does not improve the results obtained by the greedy algorithm. These results are not a surprise, since a greedy algorithm is a local search approach, it provides nearly optimal results if enough local information is available (in this case the cost of assigning a net to a given

Table 6. Comparison between a greedy algorithm and the GAHNN for a standard cost function, instances 1–12.

| Problem | Greedy | GAHNN |
|---------|--------|-------|
| 1 | 32 | 32 |
| 2 | 40 | 40 |
| 3 | 36 | 36 |
| 4 | 44 | 44 |
| 5 | 48 | 49 |
| 6 | 49 | 51 |
| 7 | 50 | 50 |
| 8 | 46 | 46 |
| 9 | 52 | 53 |
| 10 | 58 | 58 |
| 11 | 60 | 60 |
| 12 | 60 | 61 |

track). However, this greedy algorithm cannot be applied to non-standard cost function, where there is not local information available.

Finally we would like to summarize the main conclusions to be extracted from this experiments section. First, we have shown that, the metaheuristics algorithms presented in this paper are very useful when the FSCRП involves non-standard cost functions. However, in problems involving standard cost functions they do not improve the results obtained by other algorithms which uses local search information for construction solutions to the problem, such as greedy approaches. In FSCRП problems with non-standard cost functions where the computational time is not a priority, the GAHNN algorithm may provide solutions of better quality than the other approaches studied. In situations in which is needed to obtain a good solution in a few seconds, the GA with penalty function is a good election. The SAHNN algorithm has shown worse performance than the other approaches considered. We expect that a different mutation heuristic, like an ad-hoc mutation or a swapping scheme between elements of the solutions, improves its performance, however, we consider this as a possible future research.

5. Conclusions

In this paper we have presented and analyzed three meta-heuristic techniques for the FPGA segmented channel routing problem (FSCRП) with non-standard cost functions. Two of them are hybrid algorithms, based on mixing a Hopfield neural network (HNN) as a local search procedure for solving problem's constraints and two global search heuristic for improving the solutions obtained by the HNN: a Genetic Algorithm (GA) and a Simulated Annealing (SA). The third one is a GA with a penalty function to cope with the problem's constraints. Unlike previous approaches to the FSCRП, our algorithms are able to manage non-standard cost functions, in which the cost of a single assignment is not known in advanced, but it is necessary to have a full solution to the problem in order to calculate the its cost. We have described and analyzed the proposed approaches, and we have tested them in several FSCRП test instances, comparing their performances. We have found that the approach GAHNN performs better than the algorithm SAHNN and the GA with penalty function in the majority of the test functions considered. The SAHNN approach did not improve the performance of the GAHNN in any test, and only outperformed the GA with penalty function in the smallest instances considered. A comparison with a lower bound for the non-standard cost function considered has enhanced the good performance of the metaheuristics tested in this paper.

Acknowledgments

The authors really appreciate the anonymous reviewers' comments and suggestions, which have helped to improve the quality of this paper. The authors also would like to thank professor W. Banzhaf for his help with the paper.

Note

1. The real computational time measures the elapsed real time between invocation and termination of the algorithm.

References

- [1] F. N. Abuali, D. A. Schoenefeld, and R. L. Wainwright, "Terminal assignment in a communications network using genetic algorithms," in Proc. 22nd Annual ACM Computer Science Conference, J. Werth and L. Werth (eds.), ACM press, 1994, pp. 74–81.
- [2] J. Balicki, A. Stateczny, and B. Zak, "Genetic algorithms and Hopfield neural networks for solving combinatorial problems," *Appl. Math. and Comp. Sci.*, vol. 7, no. 3, pp. 568–592, 1997.
- [3] C. Bousoño-Calzón and A. R. Figueiras-Vidal, "Emerging techniques for dynamic frequency assignment: merging genetic algorithms and neural networks," in Proc. of Information Systems Technology Symposium, pp. 12.1–12.5, Aalborg, Denmark, 1998.
- [4] S. Burman, C. Kamalanathan, and N. Sherwani, "New channel segmentation model and associated routing algorithm for high performance FPGA's," in Proc. IEEE Int. Conf. Computer Aided Design, L. Trevillyan and M. Lightner (eds.), 1992, pp. 22–25.
- [5] C. Calderón-Macías, M. K. Sen, and P. L. Stoffa, "Hopfield neural networks, and mean field annealing for seismic deconvolution and multiple attenuation," *Geophysics*, vol. 62, no. 3, pp. 992–1002, 1997.
- [6] N. Funabiki, M. Yoda, J. Kitamichi, and S. Nishikawa, "A gradual neural network approach for FPGA segmented channel routing problems," *IEEE Trans. Systems, Man and Cybern., Part B*, vol. 29, pp. 481–489, 1999.
- [7] A. E. Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, and A. Mohsen, "An architecture for electrically configurable gate arrays," *IEEE J. Solid-State Circuits*, vol. 24, pp. 394–398, 1989.
- [8] A. E. Gamal, J. Greene, and V. Roychowdhury, "Segmented channel routing is nearly as efficient as channel routing (and just as hard)," in Proc. 13th Conf. Advanced Reserch VLSI, C. H. Séquin (ed.), 1991, pp. 192–211.
- [9] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley: Reading, MA, 1989.
- [10] J. González, I. Rojas, H. Pomares, M. Salmerón, and J. J. Merelo, "Web newspaper layout optimization using simulated annealing," *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 32, no. 5, pp. 686–691, 2002.
- [11] J. W. Greene, V. Roychowdhury, S. Kaptanoglu, and A. E. Gamal, "Segmented channel routing problems," in Proc. 27th Design Automat. Conf., R. C. Smith (ed.), 1990, pp. 567–572.
- [12] S. Khuri and T. Chiu, "Heuristic algorithms for the terminal assignment problem," in Proc. of the 1997 ACM Symposium on Applied Computing, B. Bryant, J. Carroll, J. Hightower and K. M. George (eds.), ACM press, 1997, pp. 247–251.
- [13] H. Kim, Y. Hayashi, and K. Nara, "An algorithm for thermal unit maintenance scheduling through combined use of GA, SA and TS," *IEEE Trans. Power Systems*, Vol. 12, no. 1, pp. 329–335, 1997.
- [14] S. Kirkpatrick, "Optimization by simulated annealing—Quantitative studies," *J. Stat. Phys.*, vol. 34, pp. 975–986, 1984.
- [15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
- [16] W. N. Li, "The complexity of the segmented channel routing," *IEEE Trans. Computer-Aided Desing*, vol. 14, pp. 518–523, 1993.
- [17] J. Richardson, M. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in Proc. 3rd Int. Conf. Genetic Algorithms and Applications, J. D. Schaffer (ed.), Morgan Kaufmann: California, 1989.
- [18] K. Roy, "A bounded search algorithm for segmented channel routing for FPGA's and associated channel architecture issues," *IEEE Trans. Computer-Aided Desing*, vol. 12, pp. 1695–1705, 1993.
- [19] V. Roychowdhury, J. W. Greene, and A. E. Gamal, "Segmented channel routing," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 75–95, 1993.
- [20] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 284–294, 2000.
- [21] S. Salcedo-Sanz, C. Bousoño-Calzón, and A. R. Figueiras-Vidal, "A mixed neural-genetic algorithm for the broadcast scheduling problem," *IEEE Trans. Wireless Commun.*, vol. 2, no. 2, pp. 277–283, 2003.

- [22] S. Salcedo-Sanz, R. Santiago-Mozos, and C. Bousoño-Calzón, "A hybrid Hopfield network-simulated annealing approach for frequency assignment in satellite communications systems," *IEEE Trans. Syst. Man Cybern. B*, vol. 34, no. 2, pp. 1108–1116, 2004.
- [23] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 2nd edition, Kluwer: Norwell, MA, 1995.
- [24] Y. Shrivastava, S. Dasgupta, and S. M. Reddy, "Guaranteed convergence in a class of Hopfield networks," *IEEE Trans. Neural Networks*, vol. 3, no. 6, pp. 951–961, 1992.
- [25] K. Smith and M. Palaniswami, and M. Krishnamoorthy, "Neural techniques for combinatorial optimization with applications," *IEEE Trans. Neural Networks*, vol. 9, no. 6, pp. 1301–1318, 1998.
- [26] G. Wang and N. Ansari, "Optimal broadcast scheduling in packet radio networks using mean field annealing," *IEEE J. Select. Areas Commun.*, vol. 15, no. 2, pp. 250–259, 1997.
- [27] Y. Watanabe, N. Mizuguchi, and Y. Fujii, "Solving optimization problems by using a Hopfield neural network and genetic algorithm combination". *Systems and Computers in Japan*, vol. 29, no. 10, pp. 68–73, 1998.
- [28] X. Yao, "Optimization by genetic annealing," in *Proc. of the 2nd Australian Conf. on Neural Networks*, M. Jabri (Ed.) Sydney, 1991, pp. 94–97.